Reducing Memory Requirements for High-Performance and Numerically Stable Gaussian Elimination

David Boland
Monash University

MONASH University

Less Memory for Energy-efficient and *Actually Useful* Gaussian Elimination

David Boland
Monash University

GROUP OF EIGHT AUSTRALIA

There are two kinds of people (in this room):

- Those who passionately care about anything to do with Gaussian Elimination
  - You will get:
    - A walk through of various different structures to perform GE.
    - Tradeoffs of parallelism, memory, pipelining, numerical stability,…
    - Disclaimer: Still room for improvement

There are two kinds of people (in this room):

- Those who passionately care about anything to do with Gaussian Elimination
  - You will get:
    - A walk through of various different structures to perform GE.
    - Tradeoffs of parallelism, memory, pipelining, numerical stability,…
    - Disclaimer: Still room for improvement
- Those who stumbled into this room accidentally after a coffee break

There are two kinds of people (in this room):

- Those who passionately care about anything to do with Gaussian Elimination
  - You will get:
    - A walk through of various different structures to perform GE.
    - Tradeoffs of parallelism, memory, pipelining, numerical stability,…
    - Disclaimer: Still room for improvement
- Those who stumbled into this room accidentally after a coffee break
  - If you care somewhat about parallel processing, you will get:
    - Some thoughts about how to reduce memory and I/O requirements for systolic arrays to use the whole FPGA.
    - Reminder of some algorithm that you learnt ages ago, something about solving matrices.

There are two kinds of people (in this room):

- Those who passionately care about anything to do with Gaussian Elimination
  - You will get:
    - A walk through of various different structures to perform GE.
    - Tradeoffs of parallelism, memory, pipelining, numerical stability,…
    - Disclaimer: Still room for improvement
- Those who stumbled into this room accidentally after a coffee break
  - If you care somewhat about parallel processing, you will get:
    - Some thoughts about how to reduce memory and I/O requirements for systolic arrays to use the whole FPGA.
    - Reminder of some algorithm that you learnt ages ago, something about solving matrices.
  - If you don't
    - Confusion about whether the speaker's ethnicity. Is he Australian/English/Canadian/Chinese

- A direct method to find a solution for Ax=b

$$A = \begin{bmatrix} 16 & 4 & 8 & -12 \\ 8 & 10 & 12 & -10 \\ 4 & -7 & -3 & 7 \\ -2 & -4.5 & 10.5 & 3.5 \end{bmatrix}, \qquad b = \begin{bmatrix} 4 \\ 4 \\ 11 \\ 3.5 \end{bmatrix}$$

Gaussian Elimination: A quick reminder

- First form an augmented matrix:

$$\begin{matrix} 16 & 4 & 8 & -12 & 4 \\ 8 & 10 & 12 & -10 & 4 \\ 4 & -7 & -3 & 7 & 11 \\ -2 & -4.5 & 10.5 & 3.5 & 3.5 \end{matrix}$$

Gaussian Elimination: A quick reminder

- First form an augmented matrix:
- Then perform row elimination between two rows.

$$\begin{matrix} 16 & 4 & 8 & -12 & 4 \\ 8 & 10 & 12 & -10 & 4 \\ 4 & -7 & -3 & 7 & 11 \\ -2 & -4.5 & 10.5 & 3.5 & 3.5 \end{matrix}$$

Gaussian Elimination: A quick reminder

- First form an augmented matrix:
- Then perform row elimination between two rows.

Introduce leading zero

$$\begin{matrix} 16 & 4 & 8 & -12 & 4 \\ 8 & 10 & 12 & -10 & 4 \\ 4 & -7 & -3 & 7 & 11 \\ -2 & -4.5 & 10.5 & 3.5 & 3.5 \end{matrix} \rightarrow \begin{matrix} 16 & 4 & 8 & -12 & 4 \\ \boxed{0} & 8 & 8 & -4 & 2 \\ 4 & -7 & -3 & 7 & 11 \\ -2 & -4.5 & 10.5 & 3.5 & 3.5 \end{matrix}$$

$$(Row_2 = Row_2 - {}^{1}/_{2}\ Row_1)$$

Gaussian Elimination: A quick reminder

- First form an augmented matrix:
- Repeat for all other rows in a column:

$$\begin{matrix} 16 & 4 & 8 & -12 & 4 \\ 0 & 8 & 8 & -4 & 2 \\ 4 & -7 & -3 & 7 & 11 \\ -2 & -4.5 & 10.5 & 3.5 & 3.5 \end{matrix} \rightarrow \begin{matrix} 16 & 4 & 8 & -12 & 4 \\ 0 & 8 & 8 & -4 & 2 \\ \boxed{0} & -8 & -5 & 10 & 10 \\ -2 & -4.5 & 10.5 & 3.5 & 3.5 \end{matrix}$$

$$(Row_3 = Row_3 - {}^1/_4\, Row_1)$$

Gaussian Elimination: A quick reminder

- First form an augmented matrix:
- Repeat for all other rows in a column:

$$\begin{matrix} 16 & 4 & 8 & -12 & 4 \\ 0 & 8 & 8 & -4 & 2 \\ 0 & -8 & -5 & 10 & 10 \\ -2 & -4.5 & 10.5 & 3.5 & 3.5 \end{matrix} \rightarrow \begin{matrix} 16 & 4 & 8 & -12 & 4 \\ 0 & 8 & 8 & -4 & 2 \\ 0 & -8 & -5 & 10 & 10 \\ 0 & -4 & 2.5 & 9 & 4 \end{matrix}$$

$$(Row_4 = Row_4 - {}^{1}/_{8}\ Row_1)$$

Gaussian Elimination: A quick reminder

- First form an augmented matrix:
- Repeat for all other rows in a column:

$$\begin{matrix} 16 & 4 & 8 & -12 & 4 \\ 0 & 8 & 8 & -4 & 2 \\ 0 & -8 & -5 & 10 & 10 \\ -2 & -4.5 & 10.5 & 3.5 & 3.5 \end{matrix} \rightarrow \begin{matrix} 16 & 4 & 8 & -12 & 4 \\ 0 & 8 & 8 & -4 & 2 \\ 0 & \boxed{0} & 3 & -6 & 12 \\ 0 & -4 & 2.5 & 9 & 4 \end{matrix}$$

$$(Row_3 = Row_3 + Row_2)$$

Gaussian Elimination: A quick reminder

- Eventually, an upper triangular matrix is formed:

$$\begin{matrix} 16 & 4 & 8 & -12 & 4 \\ 0 & 8 & 8 & -4 & 2 \\ 0 & 0 & 3 & 6 & 12 \\ 0 & 0 & 0 & 4 & -1 \end{matrix}$$

Gaussian Elimination: A quick reminder

- Eventually, an upper triangular matrix is formed:

$$\begin{matrix} 16 & 4 & 8 & -12 & 4 \\ 0 & 8 & 8 & -4 & 2 \\ 0 & 0 & 3 & 6 & 12 \\ 0 & 0 & 0 & 4 & -1 \end{matrix}$$

- Find solution by back substitution:

$$x_4 = -\frac{1}{4}$$

Gaussian Elimination: A quick reminder

- Eventually, an upper triangular matrix is formed:

$$\begin{matrix} 16 & 4 & 8 & -12 & 4 \\ 0 & 8 & 8 & -4 & 2 \\ 0 & 0 & 3 & 6 & 12 \\ 0 & 0 & 0 & 4 & -1 \end{matrix}$$

- Find solution by back substitution:

$$x_4 = -\frac{1}{4},\ x_3 = \frac{12-6*-\left(\frac{1}{4}\right)}{3}, \ldots$$

Gaussian Elimination: A quick reminder

- Good:
  - Simple algorithm
  - Often works
- Bad:
  - Slow (limited parallelism)
  - Potentially poor numerical performance

Making it fast: Parallel GE

- Do parallel row elimination:

$$\begin{matrix} x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \end{matrix} \rightarrow \begin{matrix} x & x & x & x & x \\ 0 & x & x & x & x \\ 0 & x & x & x & x \\ 0 & x & x & x & x \end{matrix}$$

Making it fast: Parallel GE

- Do parallel row elimination:

$$\begin{matrix} x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \end{matrix} \rightarrow \begin{matrix} x & x & x & x & x \\ 0 & x & x & x & x \\ 0 & x & x & x & x \\ 0 & x & x & x & x \end{matrix} \rightarrow \begin{matrix} x & x & x & x & x \\ 0 & x & x & x & x \\ 0 & 0 & x & x & x \\ 0 & 0 & x & x & x \end{matrix}$$

Making it fast: Parallel GE

- Do parallel row elimination:

$$
\begin{matrix} x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \end{matrix}
\rightarrow
\begin{matrix} x & x & x & x & x \\ 0 & x & x & x & x \\ 0 & x & x & x & x \\ 0 & x & x & x & x \end{matrix}
\rightarrow
\begin{matrix} x & x & x & x & x \\ 0 & x & x & x & x \\ 0 & 0 & x & x & x \\ 0 & 0 & x & x & x \end{matrix}
\rightarrow
\begin{matrix} x & x & x & x & x \\ 0 & x & x & x & x \\ 0 & 0 & x & x & x \\ 0 & 0 & 0 & x & x \end{matrix}
$$

Making it fast: Parallel GE

- Do parallel row elimination:

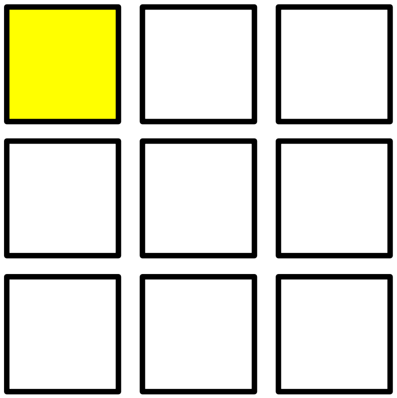
$$\begin{matrix} x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \end{matrix} \rightarrow \begin{matrix} x & x & x & x & x \\ 0 & x & x & x & x \\ 0 & x & x & x & x \\ 0 & x & x & x & x \end{matrix} \rightarrow \begin{matrix} x & x & x & x & x \\ 0 & x & x & x & x \\ 0 & 0 & x & x & x \\ 0 & 0 & x & x & x \end{matrix} \rightarrow \begin{matrix} x & x & x & x & x \\ 0 & x & x & x & x \\ 0 & 0 & x & x & x \\ 0 & 0 & 0 & x & x \end{matrix}$$

- Need parallel access to all rows
  - First need to load the entire matrix on chip
  - What about large matrices?

Making it fast: Block-based GE

- Divide matrix into blocks, load blocks into on-chip RAM

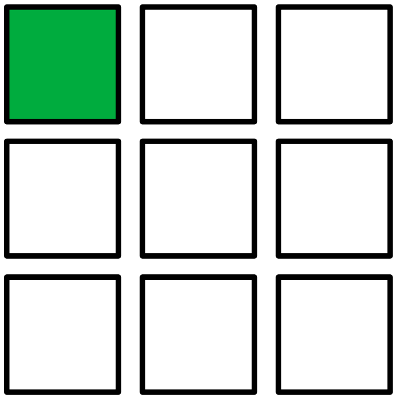
- Yellow: load to memory
- Green: update matrix
- Blue: stored matrix, needed for updates



Making it fast: Block-based GE

- Divide matrix into blocks, load blocks into on-chip RAM
- Perform parallel GE

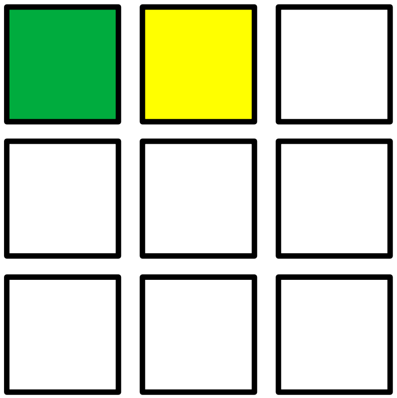
- Yellow: load to memory
- Green: update matrix
- Blue: stored matrix, needed for updates



Making it fast: Block-based GE

- Divide matrix into blocks, load blocks into on-chip RAM
- Perform parallel GE
- Double buffer for performance

- Yellow: load to memory
- Green: update matrix
- Blue: stored matrix, needed for updates

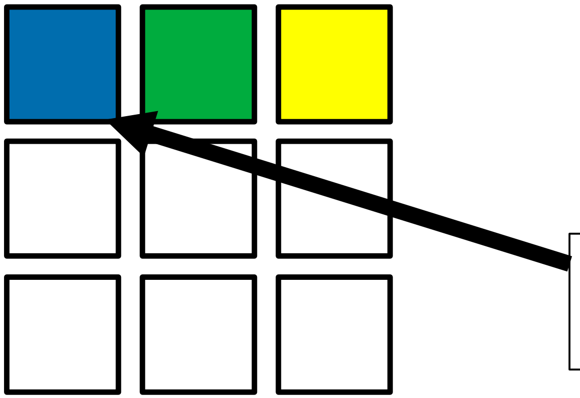


MONASH University

Making it fast: Block-based GE

- Divide matrix into blocks, load blocks into on-chip RAM
- Perform parallel GE
- Double buffer for performance
- Update to right

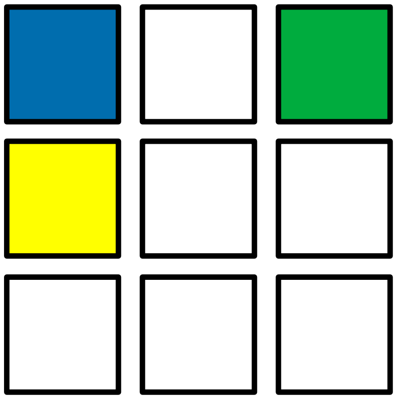
- Yellow: load to memory
- Green: update matrix
- Blue: stored matrix, needed for updates



Lower triangular matrix defining multiples to the rows for subtraction

Making it fast: Block-based GE

- Divide matrix into blocks, load blocks into on-chip RAM
- Perform parallel GE
- Double buffer for performance
- Update to right, continue to next rows

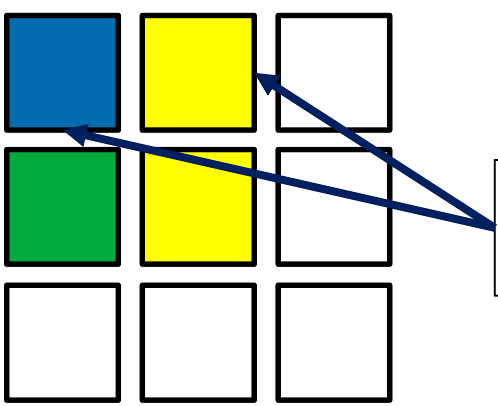


- Yellow: load to memory
- Green: update matrix
- Blue: stored matrix, needed for updates

Making it fast: Block-based GE

- Divide matrix into blocks, load blocks into on-chip RAM
- Perform parallel GE
- Double buffer for performance
- Update to right, continue to next rows

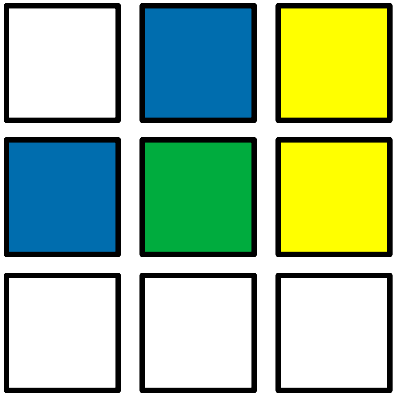
- Yellow: load to memory
- Green: update matrix
- Blue: stored matrix, needed for updates



Will subtract multiples of these rows from matrices below

Making it fast: Block-based GE

- Divide matrix into blocks, load blocks into on-chip RAM
- Perform parallel GE
- Double buffer for performance
- Update to right, continue to next rows



- Yellow: load to memory
- Green: update matrix
- Blue: stored matrix, needed for updates

Making it fast: Block-based GE

- Good:
  - Simple algorithm
  - Fast
  - Top performing FPGA implementations
- Bad:
  - Potentially poor numerical performance

Making it actually work: GE with partial pivoting

- Simple GE algorithm may fail:

$$\begin{array}{ccccc} 0 & 4 & 8 & -12 & 4 \\ 8 & 10 & 12 & -10 & 4 \\ 4 & -7 & -3 & 7 & 11 \\ -2 & -4.5 & 10.5 & 3.5 & 3.5 \end{array}$$

What multiple of Row_1 do you take from Row_2 to make this element zero?

Making it actually work: GE with partial pivoting

- Simple GE algorithm may fail:

$$\begin{matrix} 0 & 4 & 8 & -12 & 4 \\ 8 & 10 & 12 & -10 & 4 \\ 4 & -7 & -3 & 7 & 11 \\ -2 & -4.5 & 10.5 & 3.5 & 3.5 \end{matrix}$$

What multiple of Row_1 do you take from Row_2 to make this element zero?

- Just swap Row_2 and Row_1

Making it actually work: GE with partial pivoting

- More generally, for best numerical performance, you always want the row with the largest leading element unchanged
- E.g..

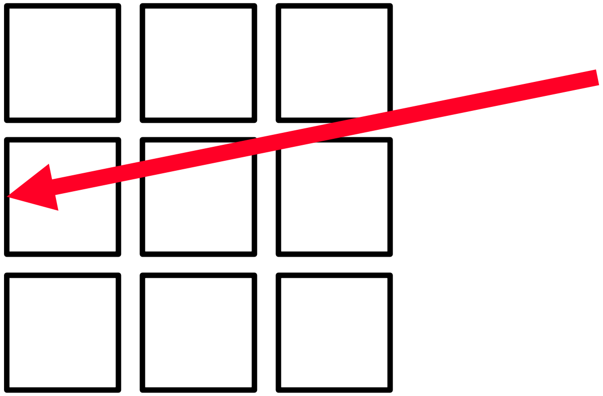
8 is largest leading element

$$\begin{matrix} 4 & 4 & 8 & -12 & 4 \\ 8 & 10 & 12 & -10 & 4 \\ 4 & -7 & -3 & 7 & 11 \\ -2 & -4.5 & 10.5 & 3.5 & 3.5 \end{matrix}$$

- Still swap Row_2 and Row_1 before elimination

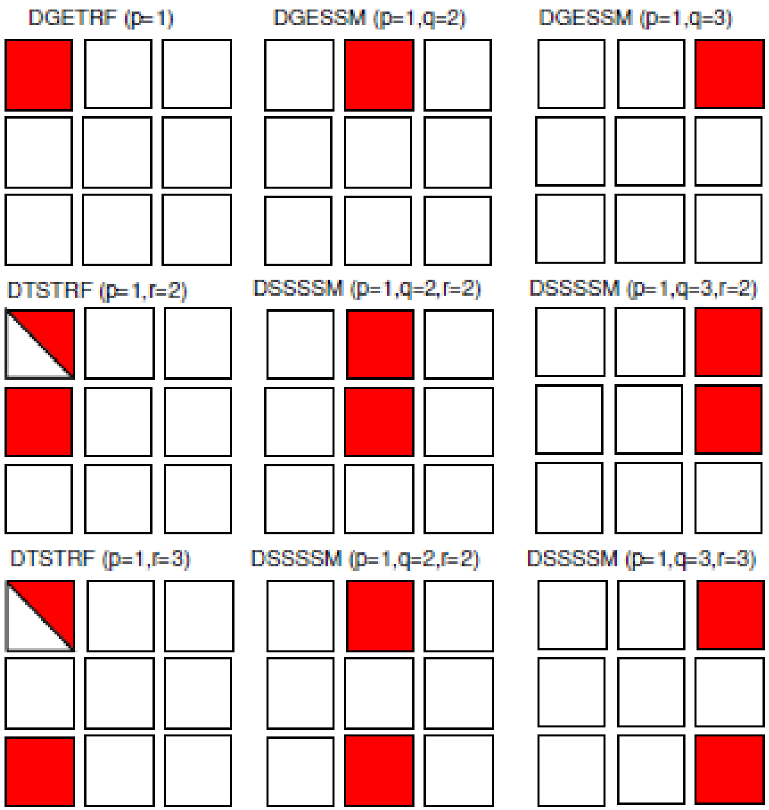
Back to the block-based algorithms

- Partial pivoting makes basic block-based Gaussian elimination difficult:



Largest leading element here…Swap between blocks?

Tiled LU Factorisation



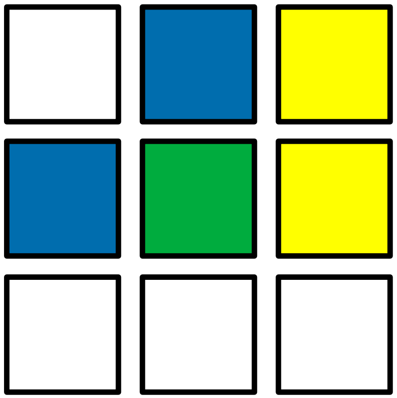
- 4 different subroutines, implement GE with partial pivoting & swap between blocks
- No known efficient FPGA implementation

But is a new problem forming?

- Even with basic block-based Gaussian elimination, 5 NxN matrices must be stored in on-chip RAM

Making it fast: Block-based GE

- Divide matrix into blocks, load blocks into on-chip RAM
- Perform parallel GE
- Double buffer for performance
- Update to right, continue to next rows



- Yellow: load to memory
- Green: update matrix
- Blue: stored matrix, needed for updates

But is a new problem forming?

- Even with basic block-based Gaussian elimination, 5 NxN matrices must be stored in on-chip RAM
- To avoid I/O problems, only N parallel processing elements can be used. ($O(N^3)$ operations and $O(N^2)$ elements to load).

But is a new problem forming?

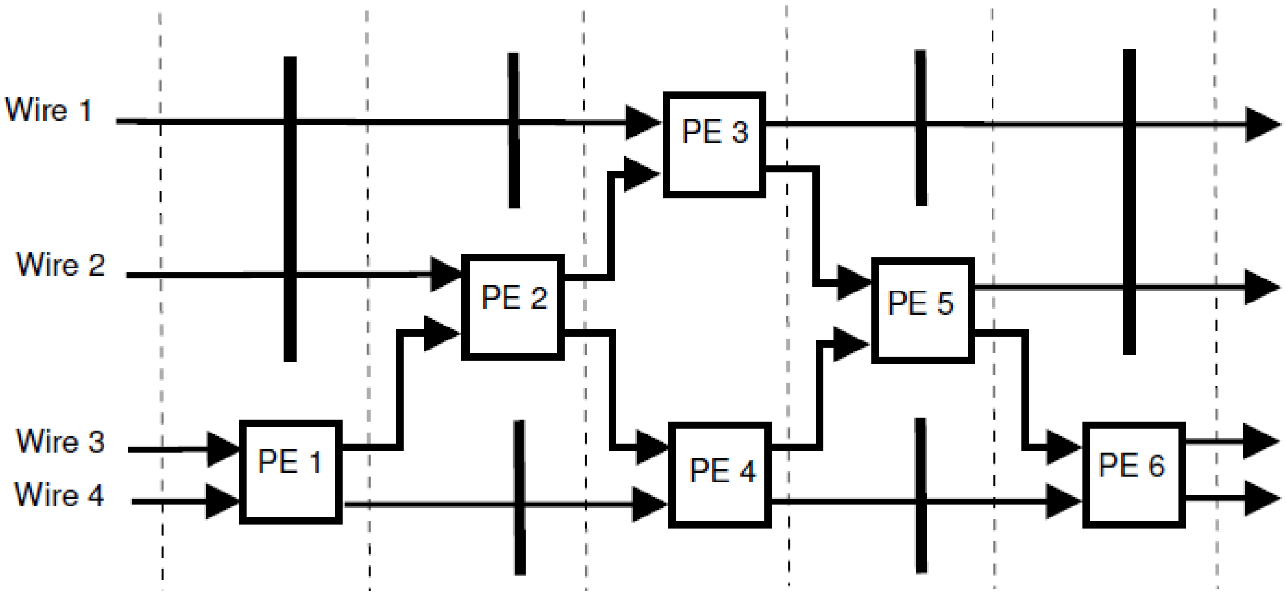
- Even with basic block-based Gaussian elimination, 5 NxN matrices must be stored in on-chip RAM
- To avoid I/O problems, only N parallel processing elements can be used. ($O(N^3)$ operations and $O(N^2)$ elements to load).
- Memory requirement ($O(N^2)$) growing faster than required number of PEs ($O(N)$)

But is a new problem forming?

- Even with basic block-based Gaussian elimination, 5 NxN matrices must be stored in on-chip RAM
- To avoid I/O problems, only N parallel processing elements can be used. ($O(N^3)$ operations and $O(N^2)$ elements to load).
- Memory requirement ($O(N^2)$) growing faster than required number of PEs ($O(N)$)
- Already a limitation on an Arria 10
  - 5 512*512 matrices use 2560 M20Ks (up to 2713 available)
  - Uses 512 PEs (up to 1688 available)

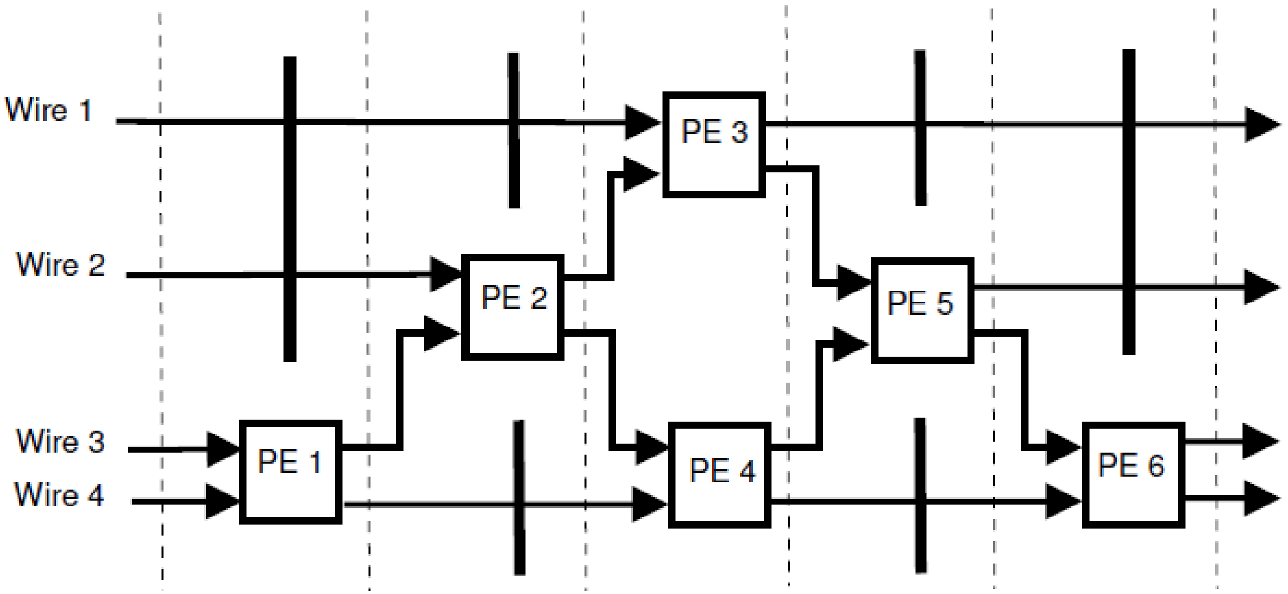
An Ingenious solution: GE with pairwise pivoting

- For numerical stability & parallelism, instead of finding largest value, find largest value between each pair of rows.



An old (pre 1990) solution: GE with pairwise pivoting

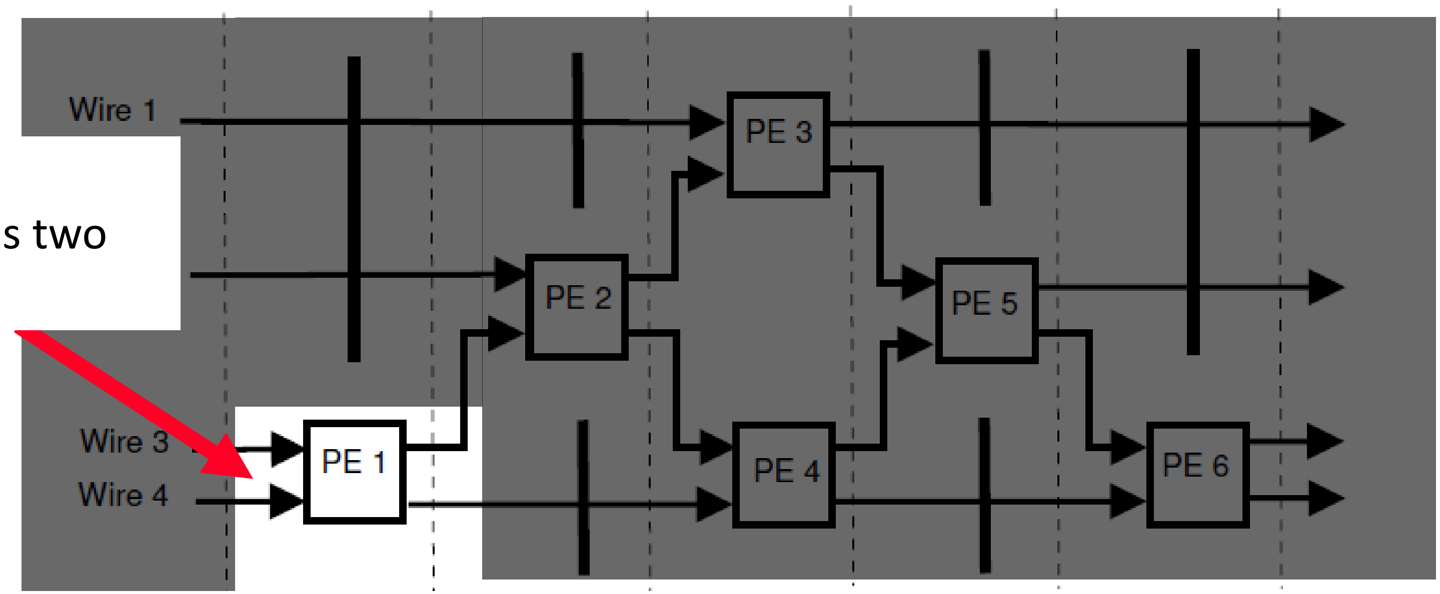
- For numerical stability & parallelism, instead of finding largest value, find largest value between each pair of rows.



An old (pre 1990) solution: GE with pairwise pivoting

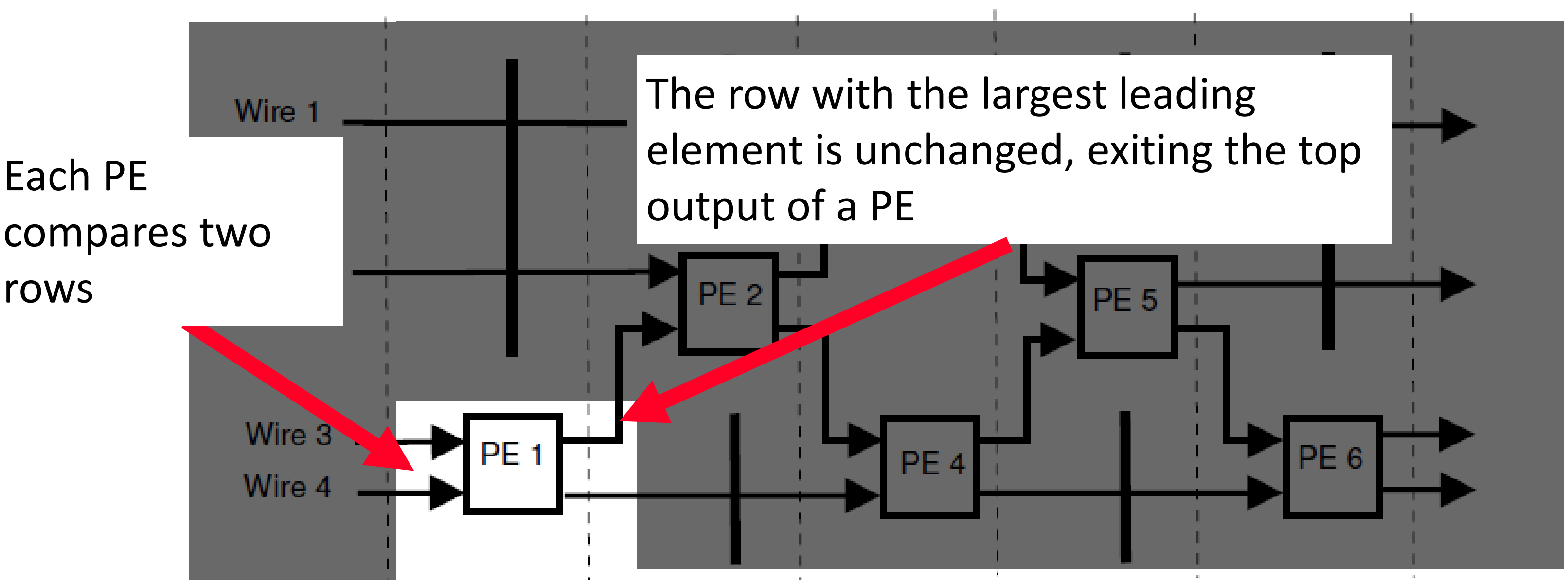
- For numerical stability & parallelism, instead of finding largest value, find largest value between each pair of rows.

Each PE compares two rows



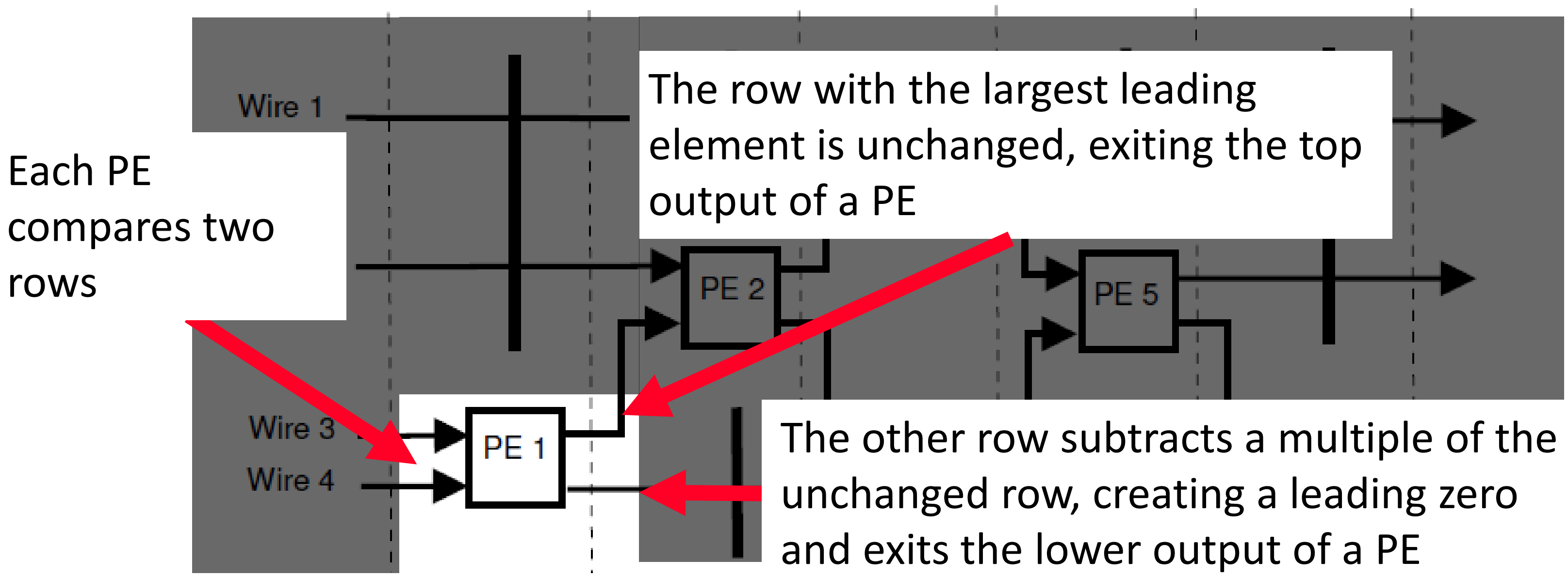
An old (pre 1990) solution: GE with pairwise pivoting

- For numerical stability & parallelism, instead of finding largest value, find largest value between each pair of rows.

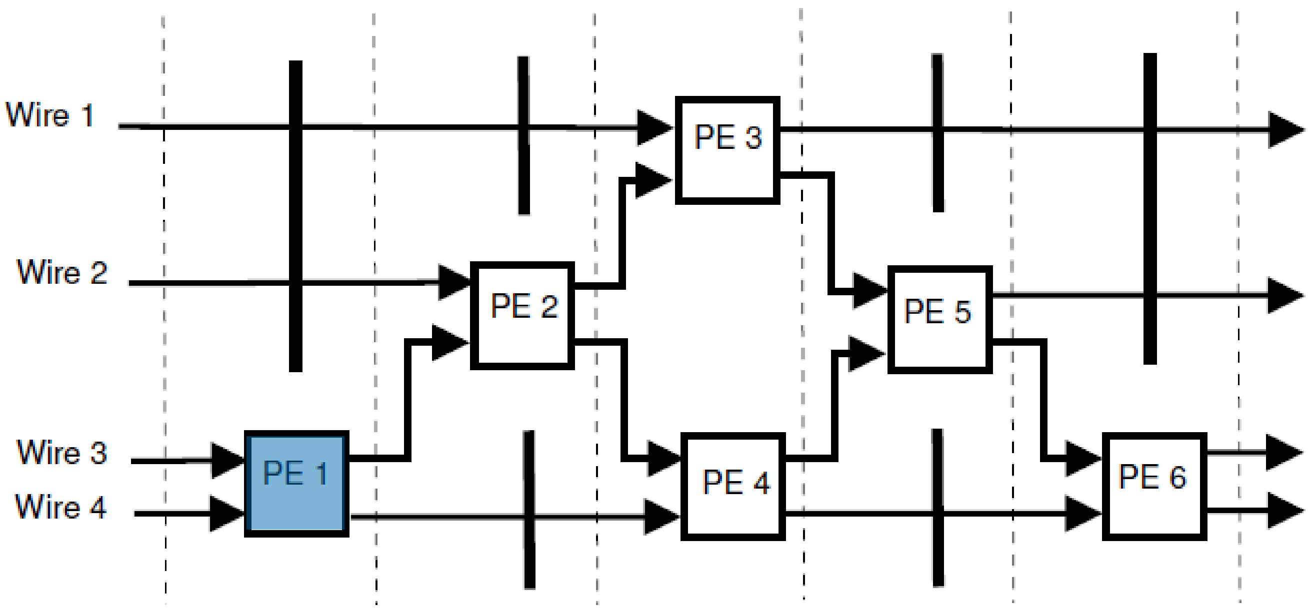


An old (pre 1990) solution: GE with pairwise pivoting

- For numerical stability & parallelism, instead of finding largest value, find largest value between each pair of rows.



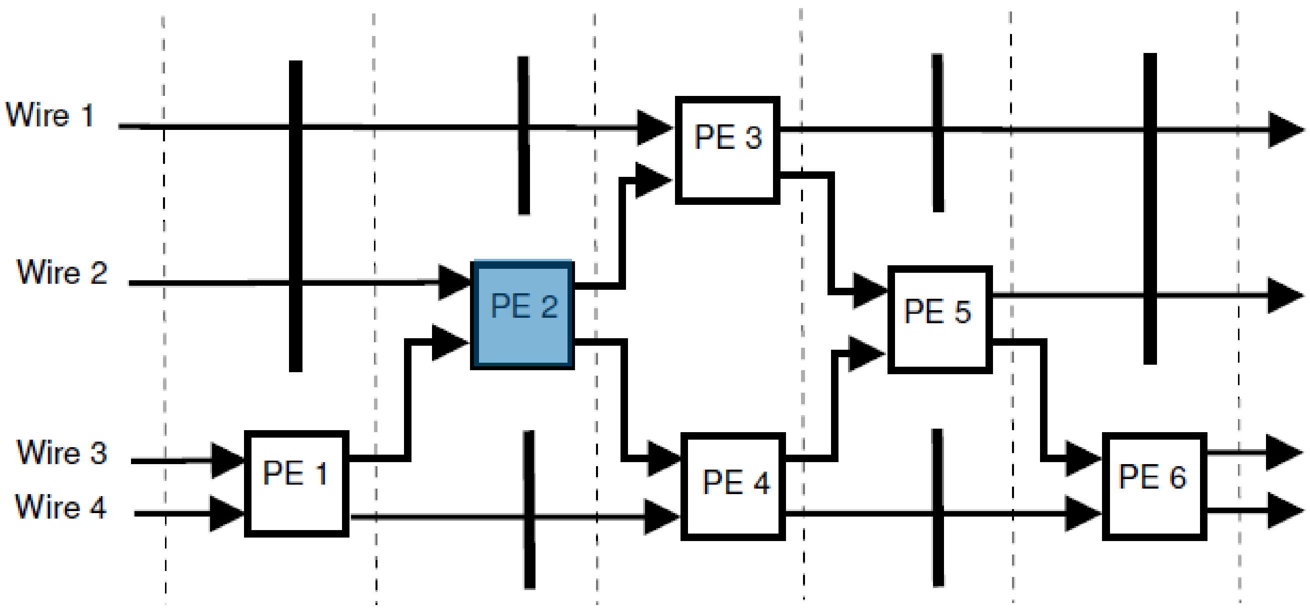
An old solution: GE with pairwise pivoting



$$\begin{matrix} x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \end{matrix} \rightarrow \begin{matrix} x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & 0 \end{matrix}$$

* Because each PE performs comparison and swap if necessary, zeros will be at same location

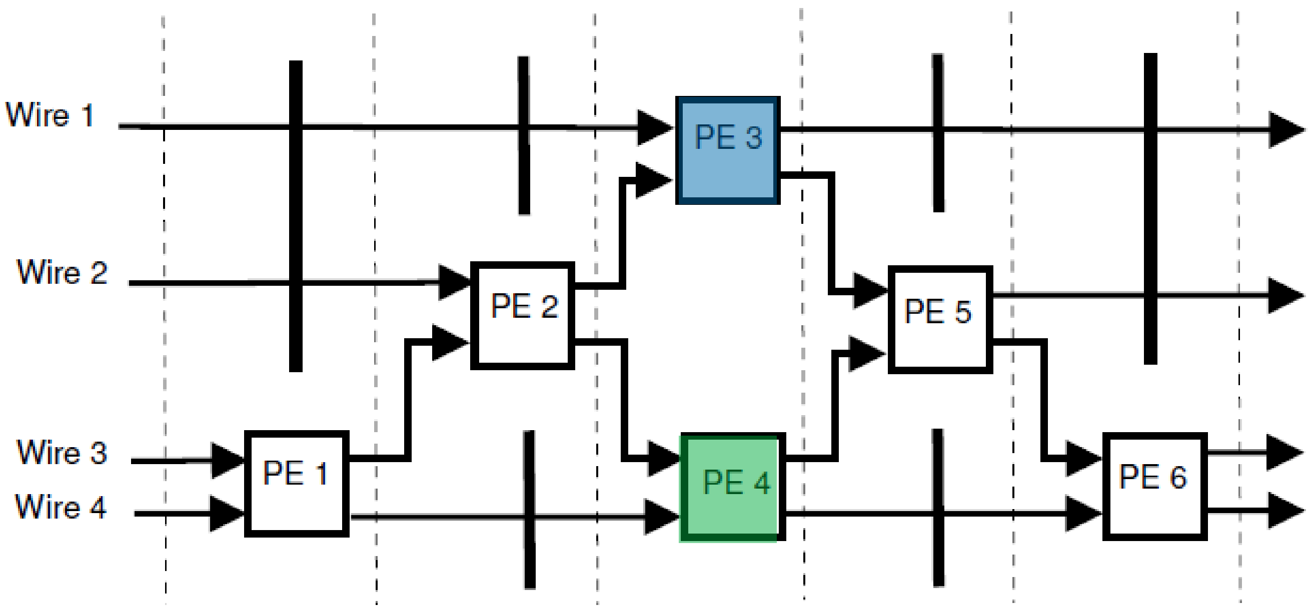
An old solution: GE with pairwise pivoting



$$\begin{matrix} x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \end{matrix} \rightarrow \begin{matrix} x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & 0 \end{matrix} \rightarrow \begin{matrix} x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & 0 \\ x & x & x & x & 0 \end{matrix}$$

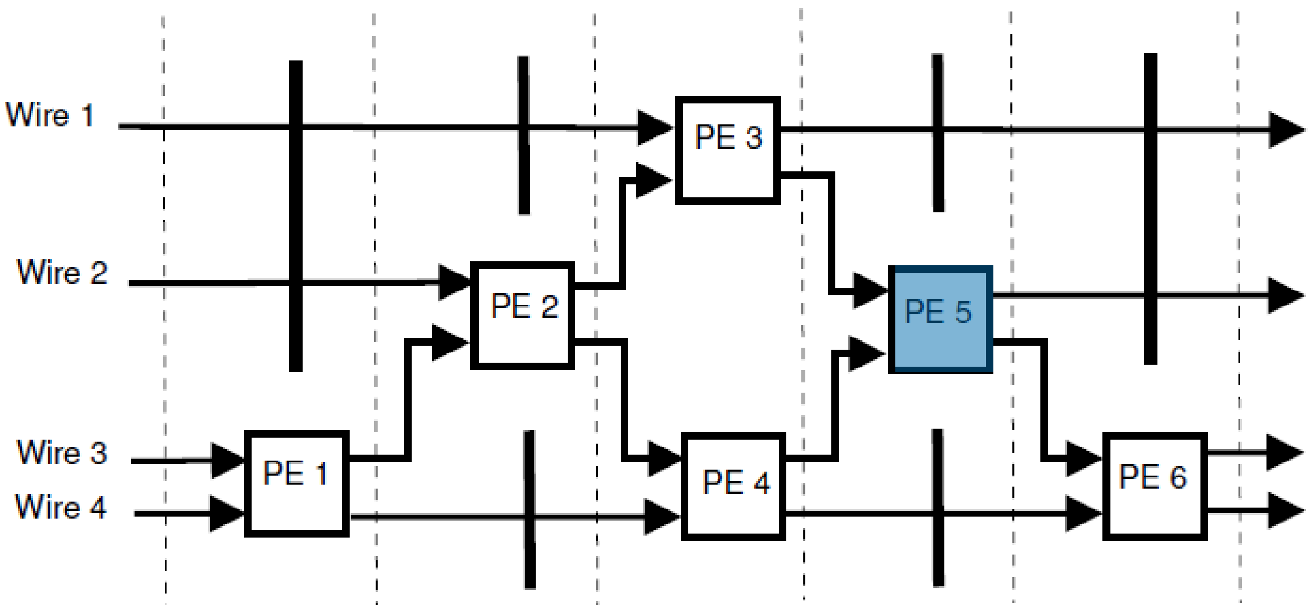
* Because each PE performs comparison and swap if necessary, zeros will be at same location

An old solution: GE with pairwise pivoting



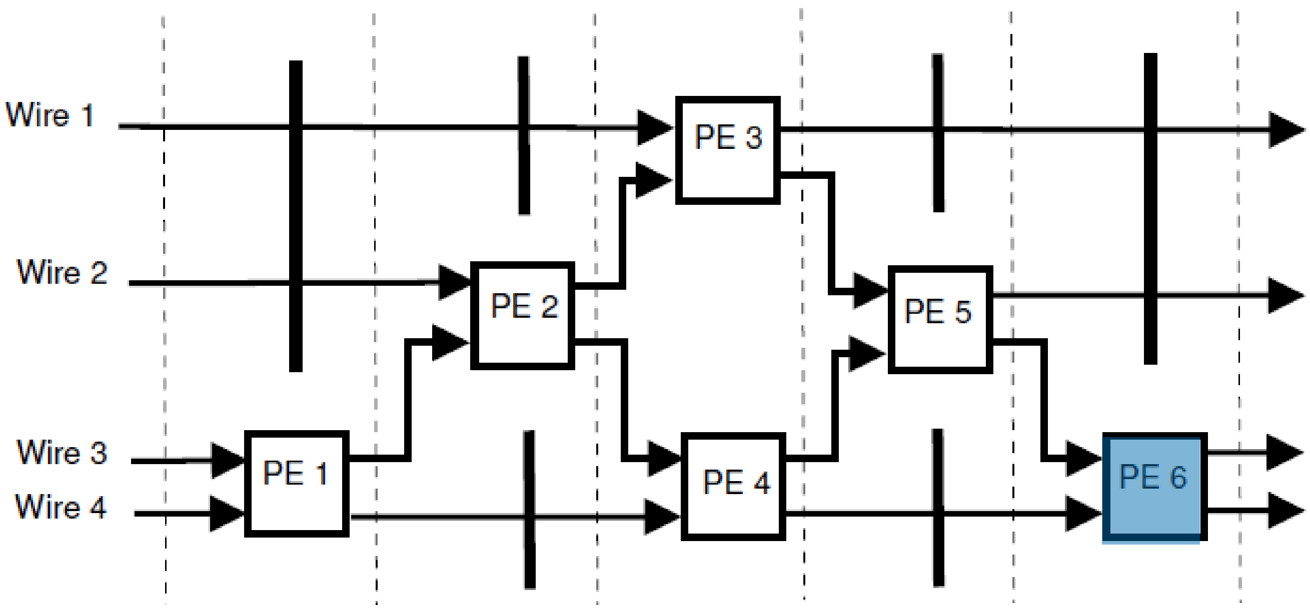
$$\begin{matrix} x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & 0 \\ x & x & x & x & 0 \end{matrix} \rightarrow \begin{matrix} x & x & x & x & x \\ x & x & x & x & 0 \\ x & x & x & x & 0 \\ x & x & x & 0 & 0 \end{matrix}$$

An old solution: GE with pairwise pivoting



$$
\begin{matrix} x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & 0 \\ x & x & x & x & 0 \end{matrix}
\rightarrow
\begin{matrix} x & x & x & x & x \\ x & x & x & x & 0 \\ x & x & x & x & 0 \\ x & x & x & 0 & 0 \end{matrix}
\rightarrow
\begin{matrix} x & x & x & x & x \\ x & x & x & x & 0 \\ x & x & x & 0 & 0 \\ x & x & x & 0 & 0 \end{matrix}
$$

An old solution: GE with pairwise pivoting

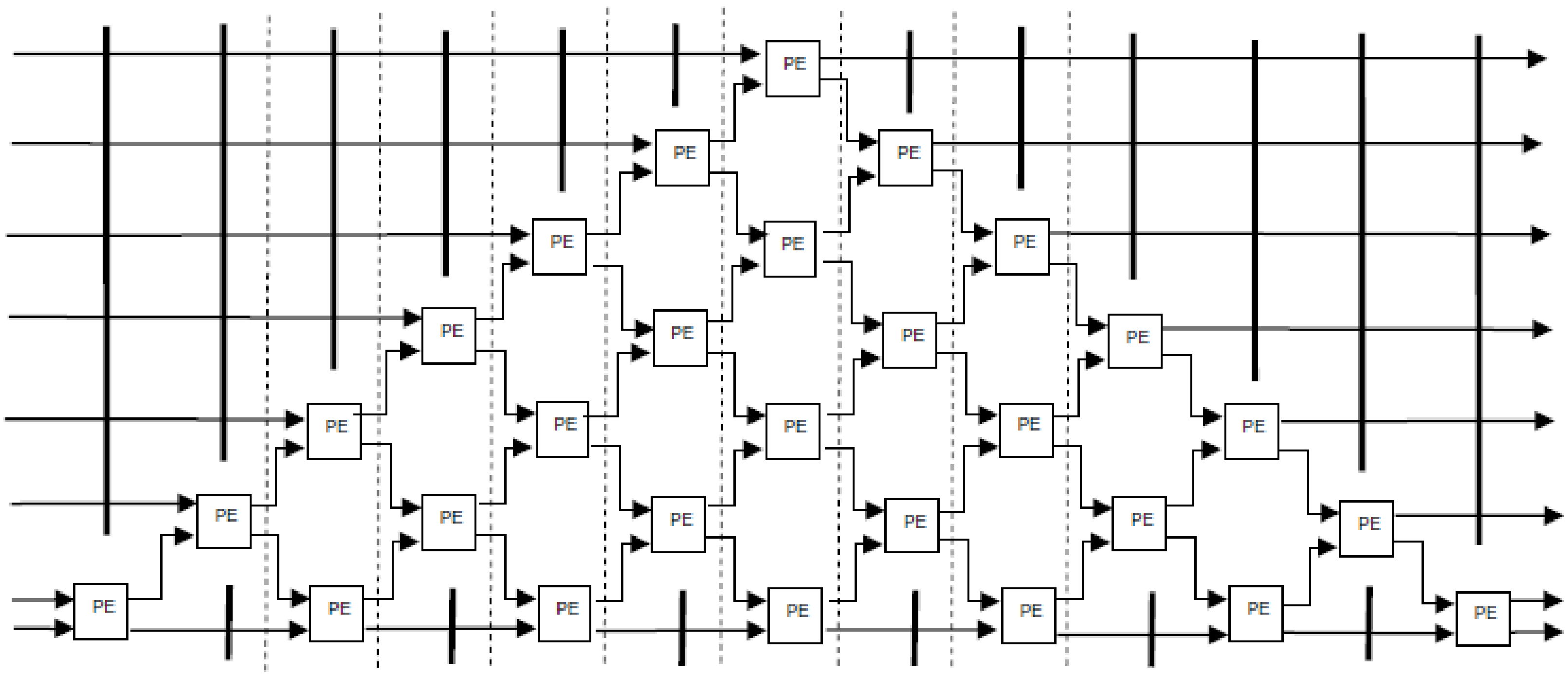


$$
\begin{matrix} x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & 0 \\ x & x & x & x & 0 \end{matrix}
\rightarrow
\begin{matrix} x & x & x & x & x \\ x & x & x & x & 0 \\ x & x & x & x & 0 \\ x & x & x & 0 & 0 \end{matrix}
\rightarrow
\begin{matrix} x & x & x & x & x \\ x & x & x & x & 0 \\ x & x & x & 0 & 0 \\ x & x & x & 0 & 0 \end{matrix}
\rightarrow
\begin{matrix} x & x & x & x & x \\ x & x & x & x & 0 \\ x & x & x & 0 & 0 \\ x & x & 0 & 0 & 0 \end{matrix}
$$

An old solution: GE with pairwise pivoting

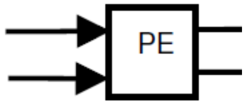
- Good:
  - Fast
  - Numerically stable
  - Advantages of systolic array (not continual access to memory)
- Bad:
  - I/O problem for large matrices

Double the matrix size



Using one PE to emulate two

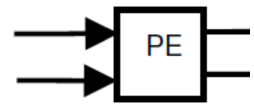
- Let’s study the output of one PE with a different input order:



$$
\begin{matrix} x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \end{matrix}
\rightarrow
\begin{matrix} x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & 0 \end{matrix}
$$

Using one PE to emulate two

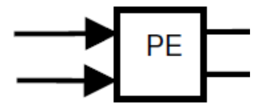
- Let’s study the output of one PE with a different input order:



$$\begin{matrix} x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \end{matrix} \rightarrow \begin{matrix} x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & 0 \end{matrix} \rightarrow \begin{matrix} x & x & x & x & x \\ x & x & x & x & 0 \\ x & x & x & x & x \\ x & x & x & x & 0 \end{matrix}$$

Using one PE to emulate two

- Let's study the output of one PE with a different input order:

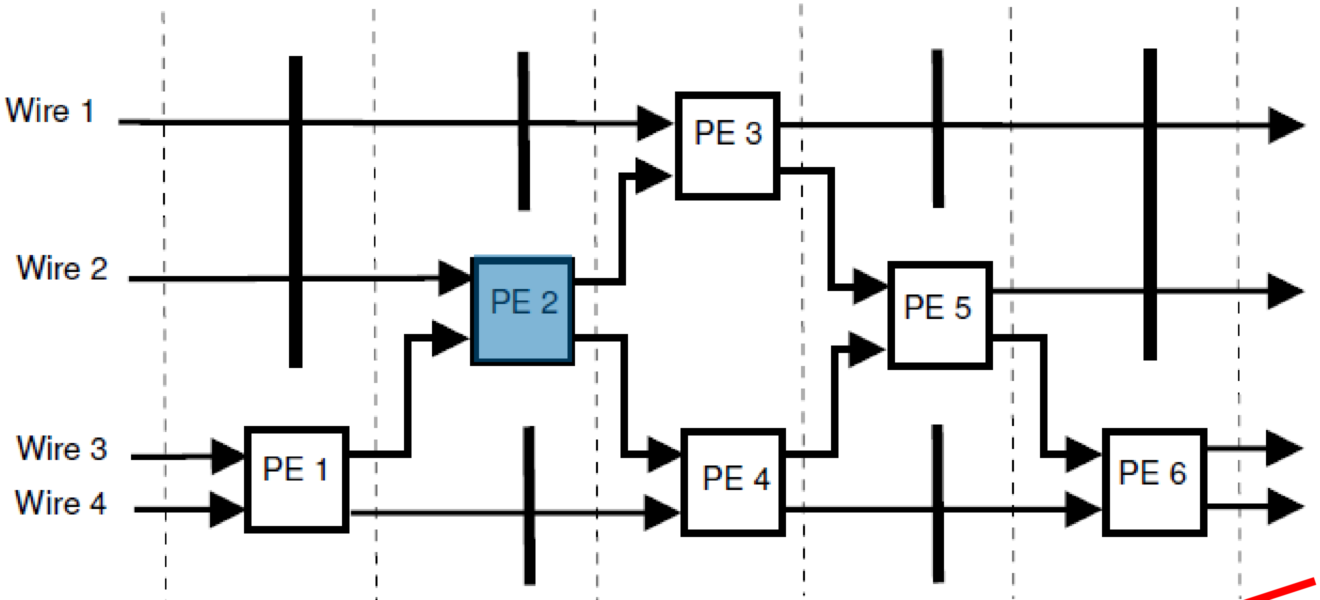


$$\begin{matrix} x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \end{matrix} \rightarrow \begin{matrix} x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & 0 \end{matrix} \rightarrow \begin{matrix} x & x & x & x & x \\ x & x & x & x & 0 \\ x & x & x & x & x \\ x & x & x & x & 0 \end{matrix}$$

- Re-arrange:

$$\begin{matrix} x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & 0 \\ x & x & x & x & 0 \end{matrix}$$

An old solution: GE with pairwise pivoting

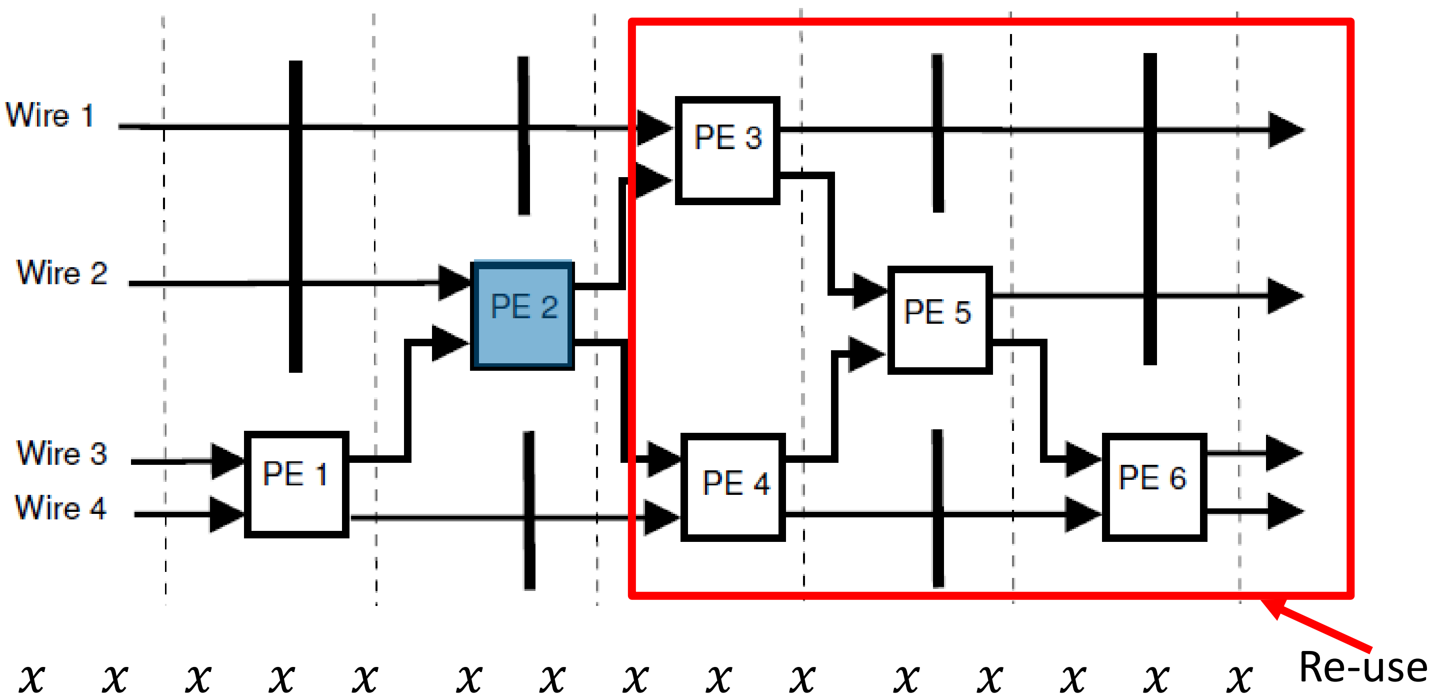


$$\begin{matrix} x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \end{matrix} \rightarrow \begin{matrix} x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & 0 \end{matrix} \rightarrow \boxed{\begin{matrix} x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & 0 \\ x & x & x & x & 0 \end{matrix}}$$

Same format

* Because each PE performs comparison and swap if necessary, zeros will be at same location

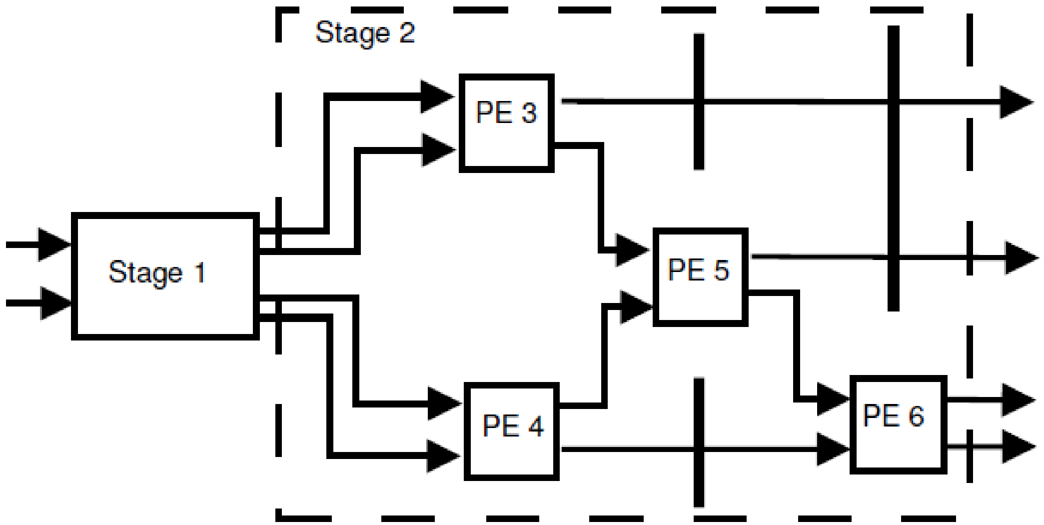
An old solution: GE with pairwise pivoting



$$
\begin{matrix} x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \end{matrix}
\rightarrow
\begin{matrix} x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & 0 \end{matrix}
\rightarrow
\begin{matrix} x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & 0 \\ x & x & x & x & 0 \end{matrix}
$$

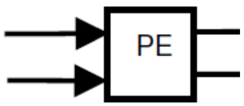
* Because each PE performs comparison and swap if necessary, zeros will be at same location

Can use second half of circuit for final solution



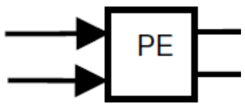
$$\begin{matrix} x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & 0 \\ x & x & x & x & 0 \end{matrix} \rightarrow \begin{matrix} x & x & x & x & x \\ x & x & x & x & 0 \\ x & x & x & x & 0 \\ x & x & x & 0 & 0 \end{matrix} \rightarrow \begin{matrix} x & x & x & x & x \\ x & x & x & x & 0 \\ x & x & x & 0 & 0 \\ x & x & x & 0 & 0 \end{matrix} \rightarrow \begin{matrix} x & x & x & x & x \\ x & x & x & x & 0 \\ x & x & x & 0 & 0 \\ x & x & 0 & 0 & 0 \end{matrix}$$

Rinse & repeat



$$
\begin{array}{ccccc}
x & x & x & x & x \\
x & x & x & x & x \\
x & x & x & x & x \\
x & x & x & x & x
\end{array}
\quad
\begin{array}{ccccc}
x & x & x & x & x \\
x & x & x & x & 0 \\
x & x & x & x & x \\
x & x & x & x & 0
\end{array}
$$

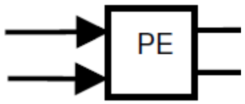
Rinse & repeat



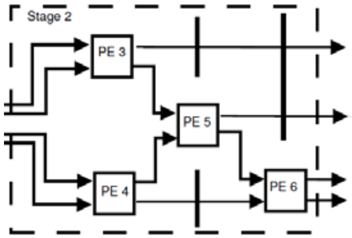
Re-arrange

$$
\begin{matrix} x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \end{matrix}
\quad
\begin{matrix} x & x & x & x & x \\ x & x & x & x & 0 \\ x & x & x & x & x \\ x & x & x & x & 0 \end{matrix}
\quad
\begin{matrix} x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & 0 \\ x & x & x & x & 0 \end{matrix}
$$

Rinse & repeat

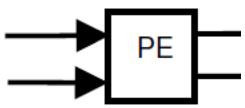


Re-arrange

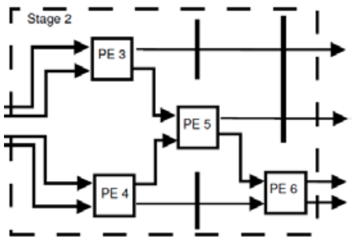


$$
\begin{matrix} x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \end{matrix}
\quad
\begin{matrix} x & x & x & x & x \\ x & x & x & x & 0 \\ x & x & x & x & x \\ x & x & x & x & 0 \end{matrix}
\quad
\begin{matrix} x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & 0 \\ x & x & x & x & 0 \end{matrix}
\quad
\begin{matrix} x & x & x & x & x \\ x & x & x & x & 0 \\ x & x & x & 0 & 0 \\ x & x & 0 & 0 & 0 \end{matrix}
$$

Rinse & repeat



Re-arrange



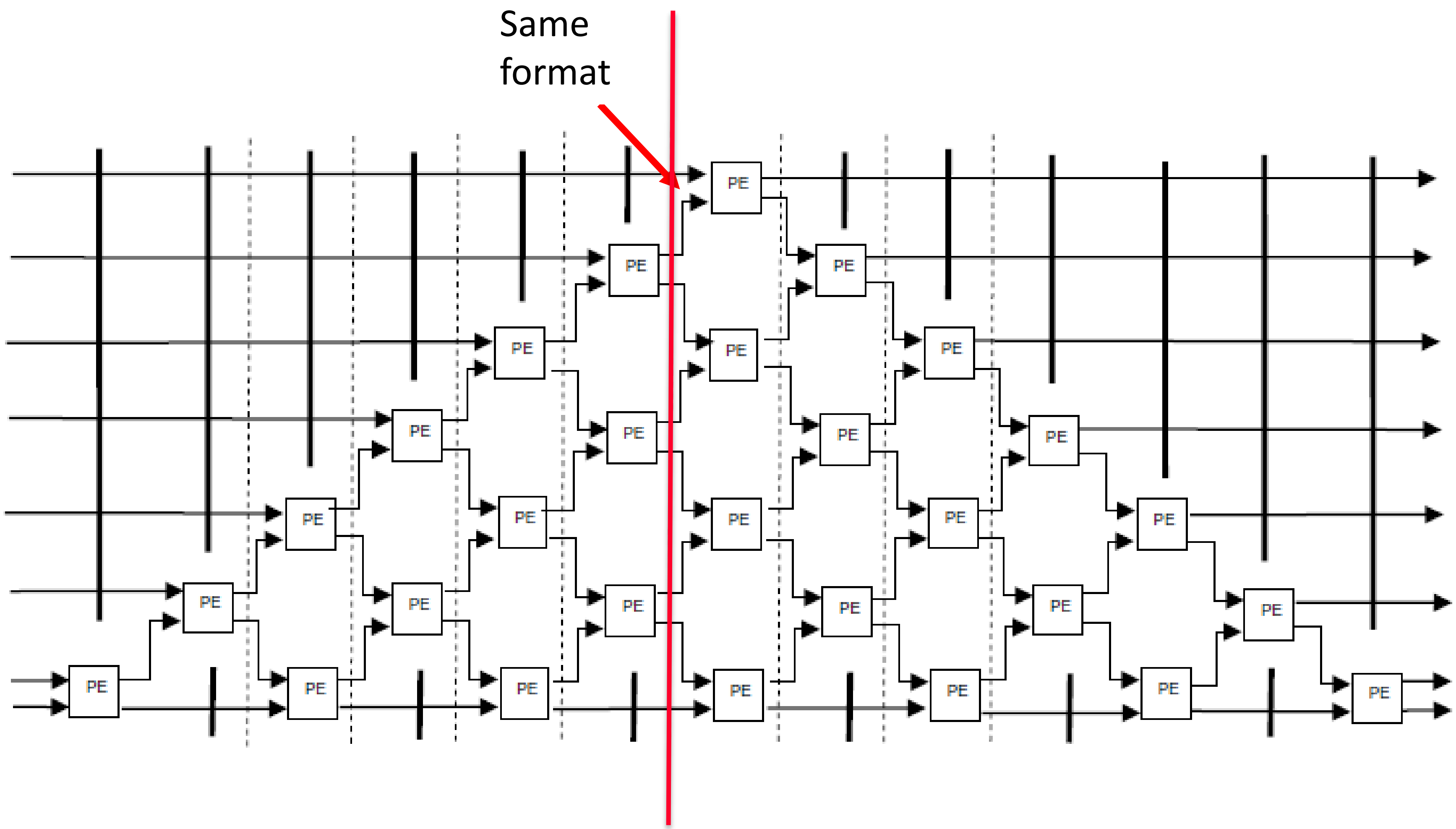
$$
\begin{matrix} x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \end{matrix}
\rightarrow
\begin{matrix} x & x & x & x & x \\ x & x & x & x & 0 \\ x & x & x & x & x \\ x & x & x & x & 0 \\ x & x & x & x & x \\ x & x & x & x & 0 \\ x & x & x & x & x \\ x & x & x & x & 0 \end{matrix}
\rightarrow
\begin{matrix} x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & 0 \\ x & x & x & x & 0 \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & 0 \\ x & x & x & x & 0 \end{matrix}
\rightarrow
\begin{matrix} x & x & x & x & x \\ x & x & x & x & 0 \\ x & x & x & 0 & 0 \\ x & x & 0 & 0 & 0 \\ x & x & x & x & x \\ x & x & x & x & 0 \\ x & x & x & 0 & 0 \\ x & x & 0 & 0 & 0 \end{matrix}
$$

Rinse & repeat

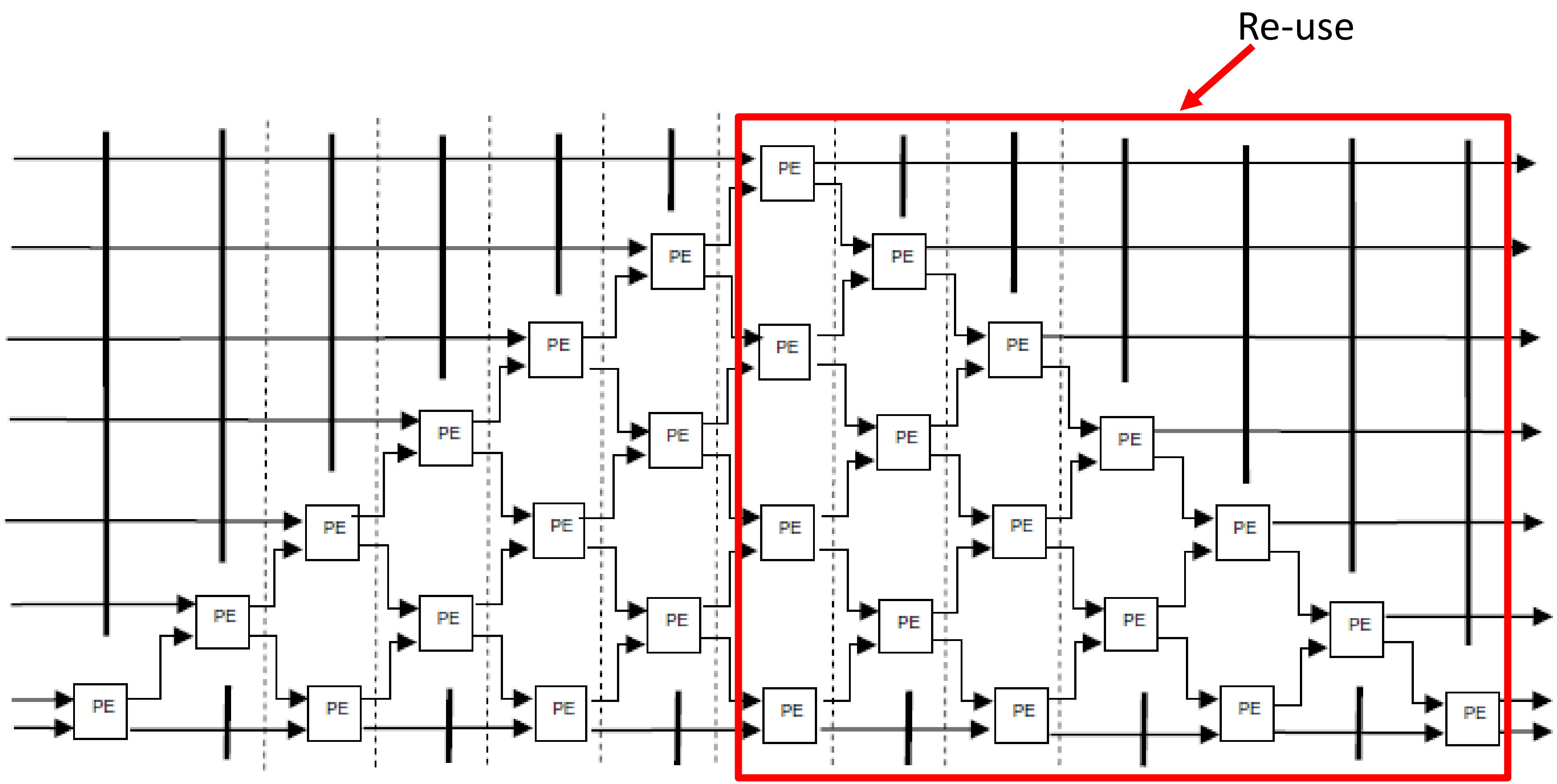
Re-arrange

- $\begin{matrix} x & x & x & x & x \\ x & x & x & x & 0 \\ x & x & x & 0 & 0 \\ x & x & 0 & 0 & 0 \\ x & x & x & x & x \\ x & x & x & x & 0 \\ x & x & x & 0 & 0 \\ x & x & 0 & 0 & 0 \end{matrix} \rightarrow \begin{matrix} x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & 0 \\ x & x & x & x & 0 \\ x & x & x & 0 & 0 \\ x & x & x & 0 & 0 \\ x & x & 0 & 0 & 0 \\ x & x & 0 & 0 & 0 \end{matrix}$

Double the matrix size

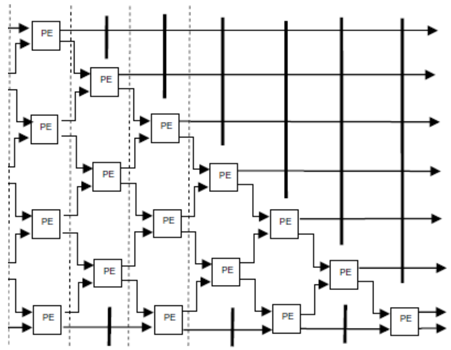


Double the matrix size



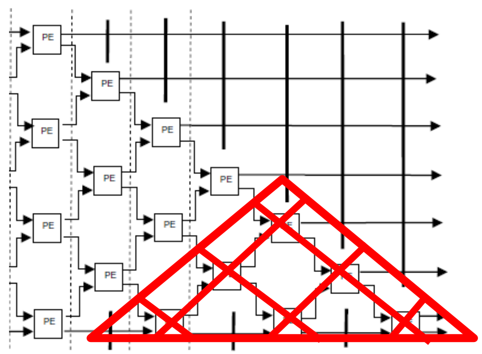
Rinse & repeat

Re-arrange



- $\begin{matrix} x & x & x & x & x \\ x & x & x & x & 0 \\ x & x & x & 0 & 0 \\ x & x & 0 & 0 & 0 \\ x & x & x & x & x \\ x & x & x & x & 0 \\ x & x & x & 0 & 0 \\ x & x & 0 & 0 & 0 \end{matrix} \rightarrow \begin{matrix} x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & 0 \\ x & x & x & x & 0 \\ x & x & x & 0 & 0 \\ x & x & x & 0 & 0 \\ x & x & 0 & 0 & 0 \\ x & x & 0 & 0 & 0 \end{matrix} \rightarrow \begin{matrix} x & x & x & x & x & x & x & x & x \\ x & x & x & x & x & x & x & x & 0 \\ x & x & x & x & x & x & x & 0 & 0 \\ x & x & x & x & x & x & 0 & 0 & 0 \\ x & x & x & x & x & 0 & 0 & 0 & 0 \\ x & x & x & x & 0 & 0 & 0 & 0 & 0 \\ x & x & x & 0 & 0 & 0 & 0 & 0 & 0 \\ x & x & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{matrix}$

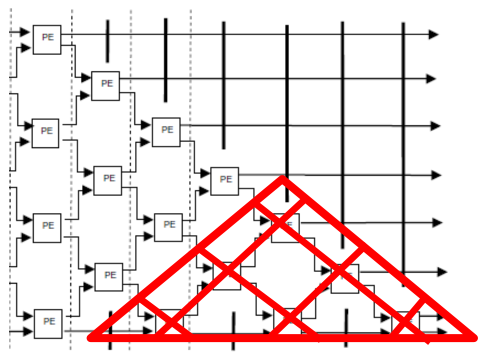
An alternative output circuit



Re-arrange

- $\begin{matrix} x & x & x & x & x \\ x & x & x & x & 0 \\ x & x & x & 0 & 0 \\ x & x & 0 & 0 & 0 \\ x & x & x & x & x \\ x & x & x & x & 0 \\ x & x & x & 0 & 0 \\ x & x & 0 & 0 & 0 \end{matrix} \rightarrow \begin{matrix} x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & 0 \\ x & x & x & x & 0 \\ x & x & x & 0 & 0 \\ x & x & x & 0 & 0 \\ x & x & 0 & 0 & 0 \\ x & x & 0 & 0 & 0 \end{matrix} \rightarrow \begin{matrix} x & x & x & x & x & x & x & x & x \\ x & x & x & x & x & x & x & x & 0 \\ x & x & x & x & x & x & x & 0 & 0 \\ x & x & x & x & x & x & 0 & 0 & 0 \\ x & x & x & x & x & 0 & 0 & 0 & 0 \\ x & x & x & x & 0 & 0 & 0 & 0 & 0 \\ x & x & x & 0 & 0 & 0 & 0 & 0 & 0 \\ x & x & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{matrix}$

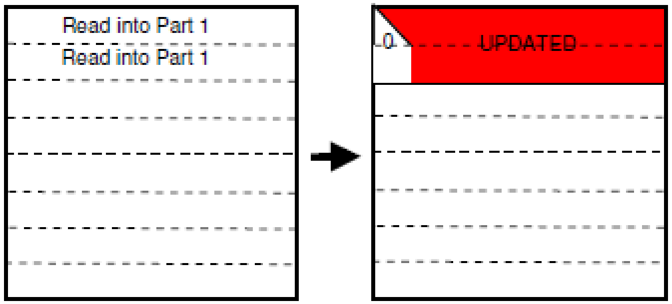
An alternative output circuit



Re-arrange

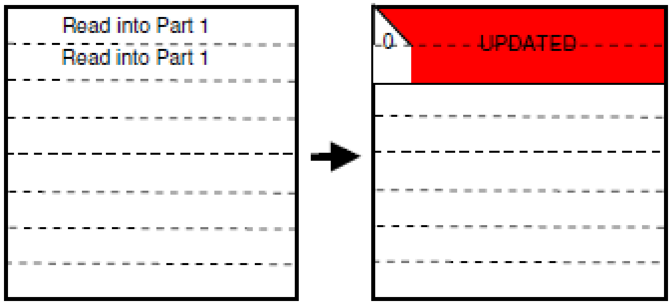
- $\begin{matrix} x & x & x & x & x \\ x & x & x & x & 0 \\ x & x & x & 0 & 0 \\ x & x & 0 & 0 & 0 \\ x & x & x & x & x \\ x & x & x & x & 0 \\ x & x & x & 0 & 0 \\ x & x & 0 & 0 & 0 \end{matrix} \rightarrow \begin{matrix} x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & 0 \\ x & x & x & x & 0 \\ x & x & x & 0 & 0 \\ x & x & x & 0 & 0 \\ x & x & 0 & 0 & 0 \\ x & x & 0 & 0 & 0 \end{matrix} \rightarrow \begin{matrix} x & x & x & x & x & x & x & x & x \\ x & x & x & x & x & x & x & x & 0 \\ x & x & x & x & x & x & x & 0 & 0 \\ x & x & x & x & x & x & 0 & 0 & 0 \\ x & x & x & x & x & 0 & 0 & 0 & 0 \\ x & x & x & x & x & 0 & 0 & 0 & 0 \\ x & x & x & x & x & 0 & 0 & 0 & 0 \\ x & x & x & x & x & 0 & 0 & 0 & 0 \end{matrix}$

Updating a large matrix

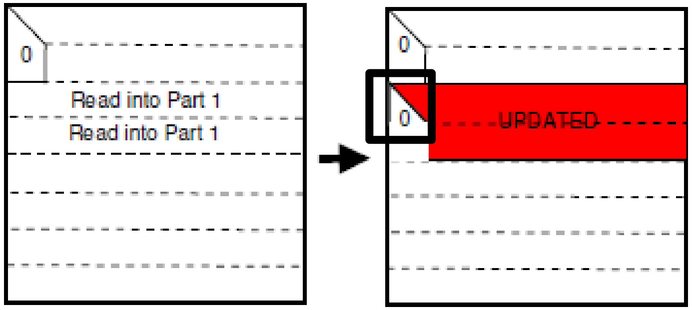


(a) First pass

Updating a large matrix

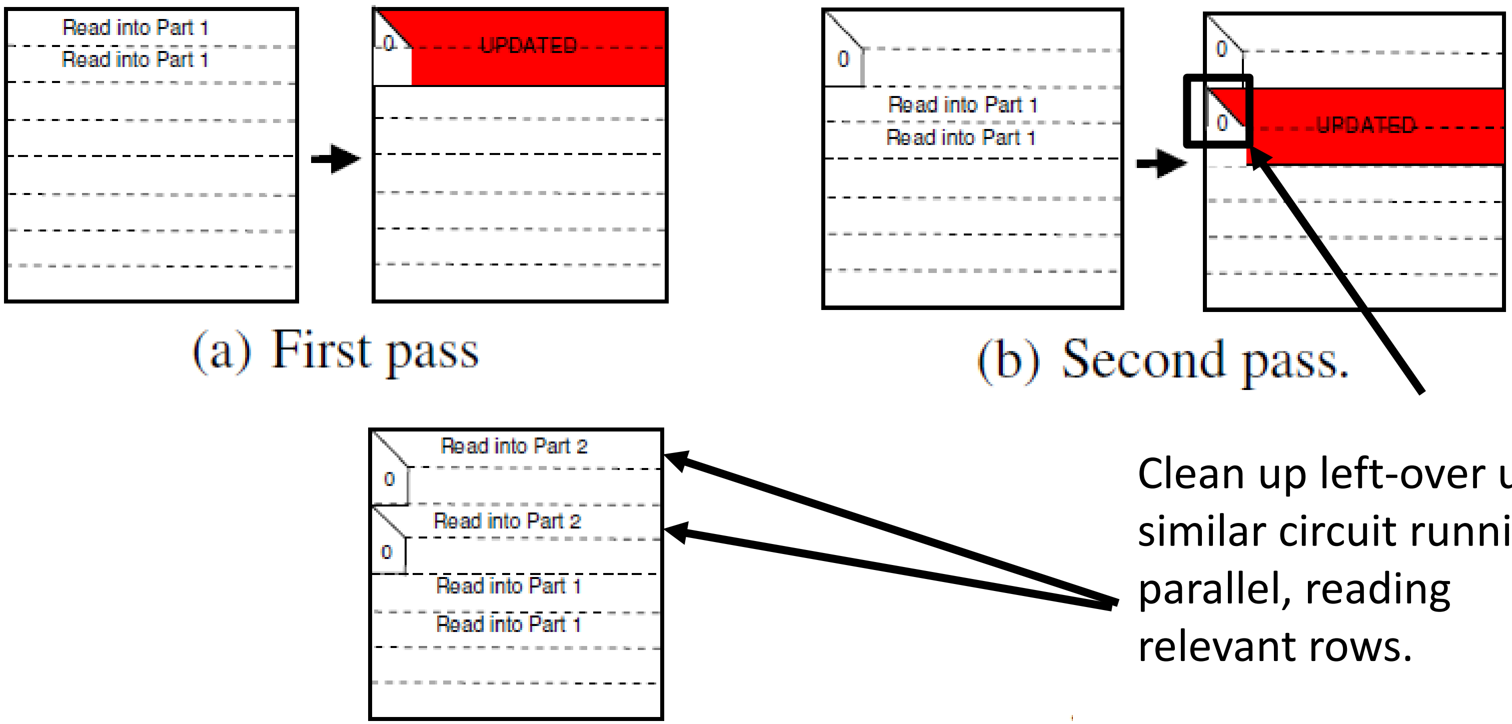


(a) First pass



(b) Second pass.

Updating a large matrix



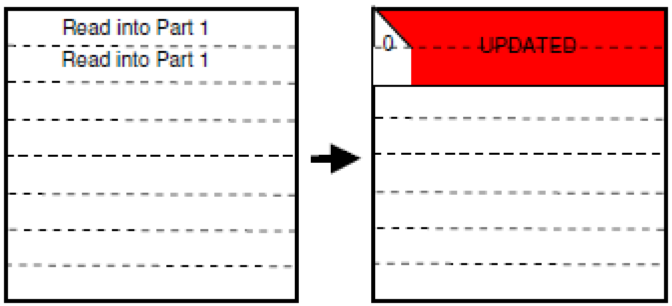
(a) First pass

(b) Second pass.

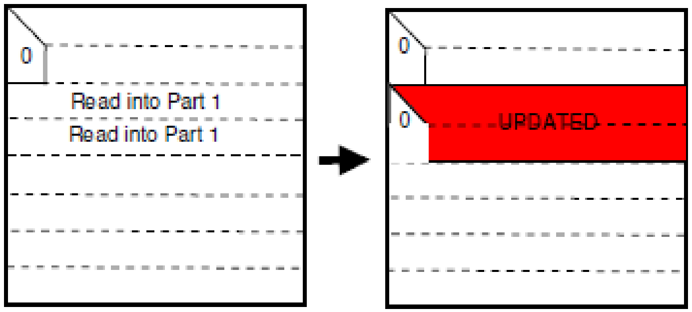
(c) Third pass.

Clean up left-over using similar circuit running in parallel, reading relevant rows.

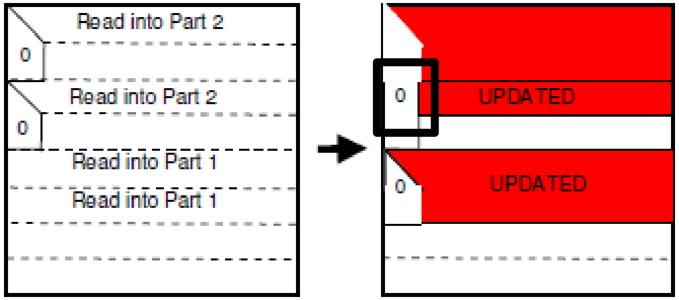
Updating a large matrix



(a) First pass



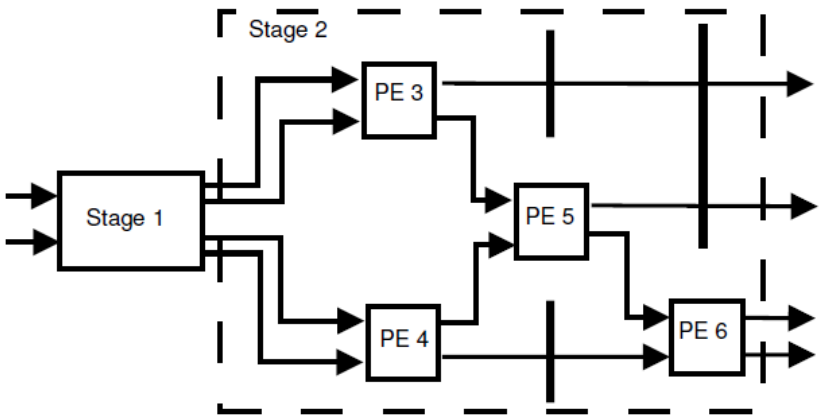
(b) Second pass.



(c) Third pass.

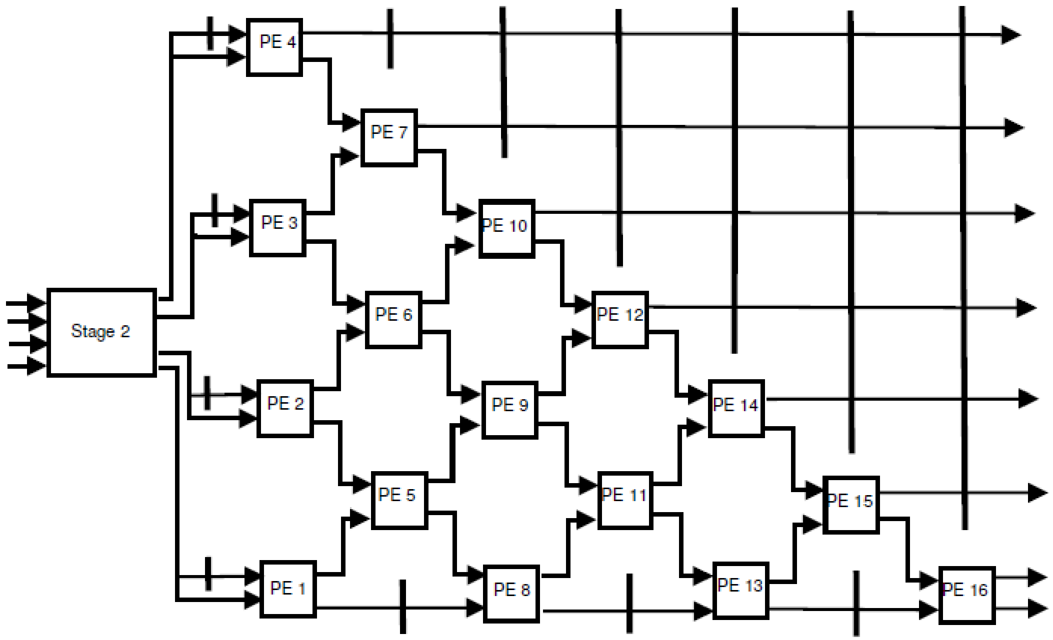
Efficiency concerns

- Second half only gets inputs every other cycle. PEs wasted



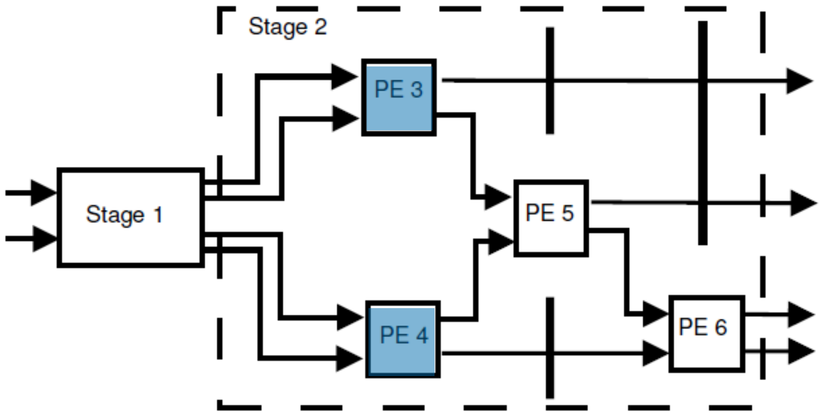
Efficiency concerns

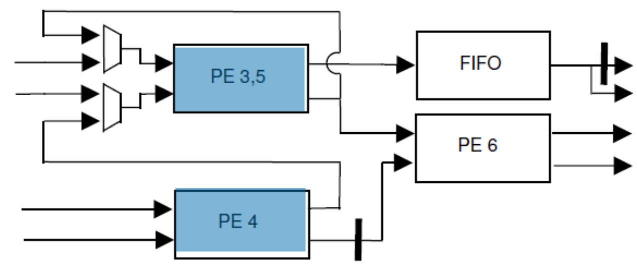
- Second half only gets inputs every four cycles. PEs wasted



Solution: share some PEs

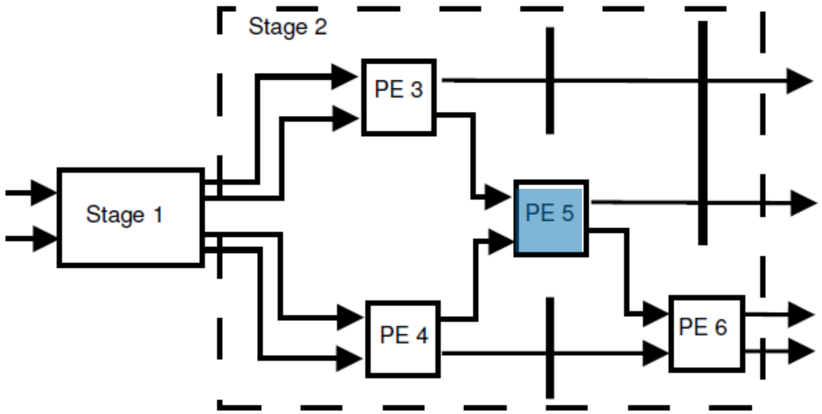
- Not 100% efficient (could be done, but trade for on-chip memory)
- Cycle 1:

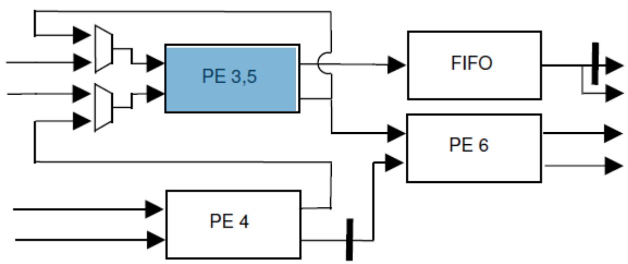




Solution: share some PEs

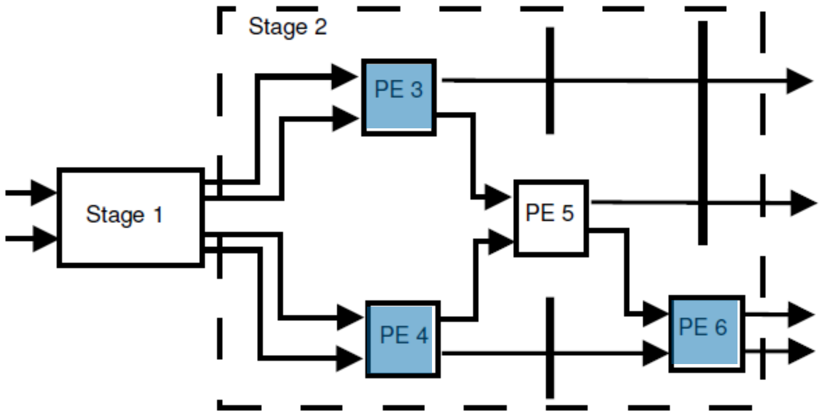
- Not 100% efficient (could be done, but trade for on-chip memory)
- Cycle 2:

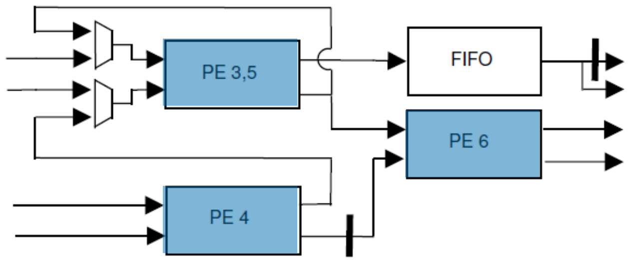




Solution: share some PEs

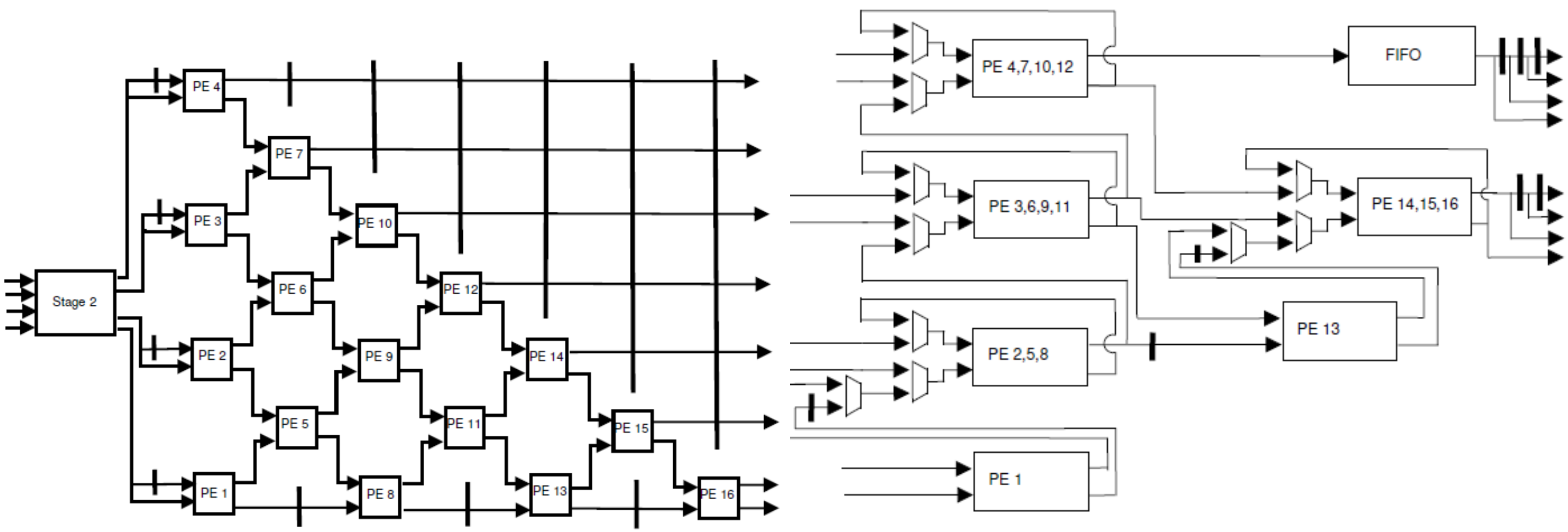
- Not 100% efficient (could be done, but trade for on-chip memory)
- Cycle 3:





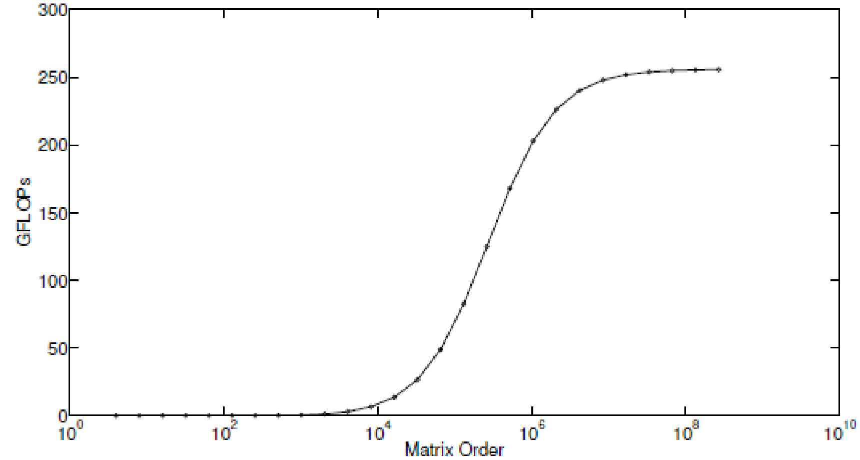
Solution: share some PEs

- Not 100% efficient (could be done, but trade for on-chip memory)



End Result (with other optimisations)

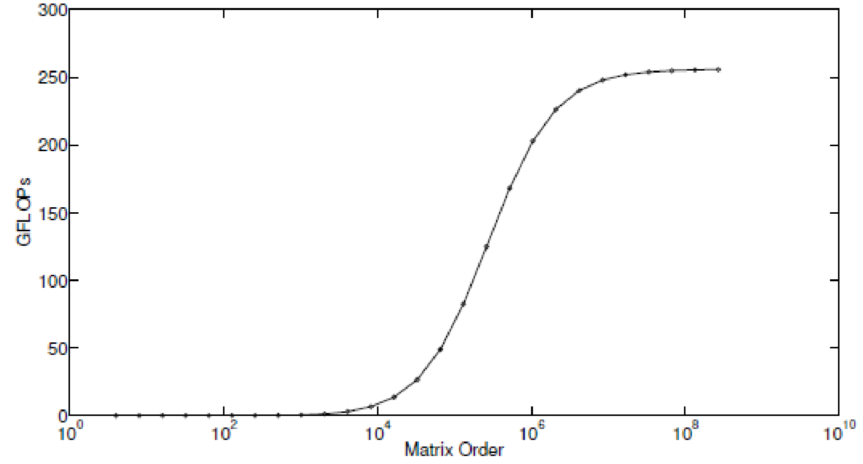
	Resource Use		
Entity	ALMs (1000s)	DSPs	M20Ks
Individual PE	0.15	1	1
Stage 1	0.9	5	4
Stage 2	2.6	11	18
Stage 3	3.6	14	27
Stage 4	5.7	20	45
Stage 5	10	32	80
Stage 6	19	56	146
Stage 7	35	104	280
Stage 8	70	200	542
Stage 9	95	260	395
Full design	338	962	1932



End Result (with other optimisations)

Limited by Slices (mainly due to pipeline registers to boost clock frequency)

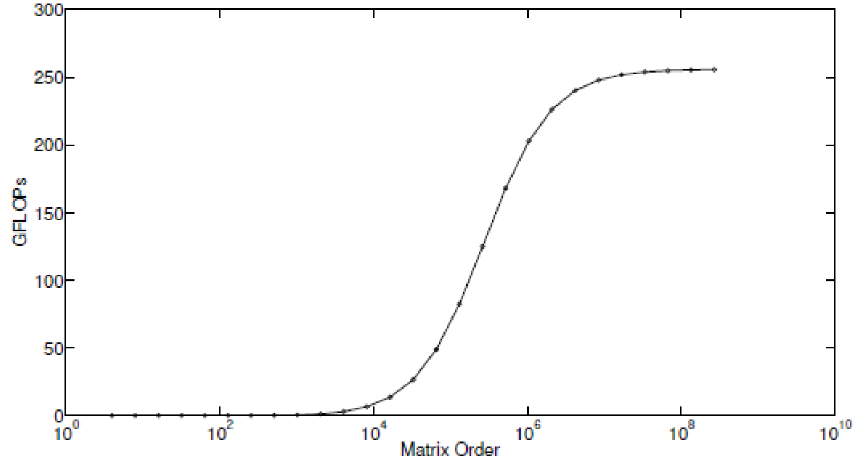
	Resource Use		
Entity	ALMs (1000s)	DSPs	M20Ks
Individual PE	0.15	1	1
Stage 1	0.9	5	4
Stage 2	2.6	11	18
Stage 3	3.6	14	27
Stage 4	5.7	20	45
Stage 5	10	32	80
Stage 6	19	56	146
Stage 7	35	104	280
Stage 8	70	200	542
Stage 9	95	260	395
Full design	338	962	1932



End Result (with other optimisations)

Limited by Slices (mainly due to pipeline registers to boost clock frequency)

	Resource Use		
Entity	ALMs (1000s)	DSPs	M20Ks
Individual PE	0.15	1	1
Stage 1	0.9	5	4
Stage 2	2.6	11	18
Stage 3	3.6	14	27
Stage 4	5.7	20	45
Stage 5	10	32	80
Stage 6	19	56	146
Stage 7	35	104	280
Stage 8	70	200	542
Stage 9	95	260	395
Full design	338	962	1932



Use more DSPs Use less memory

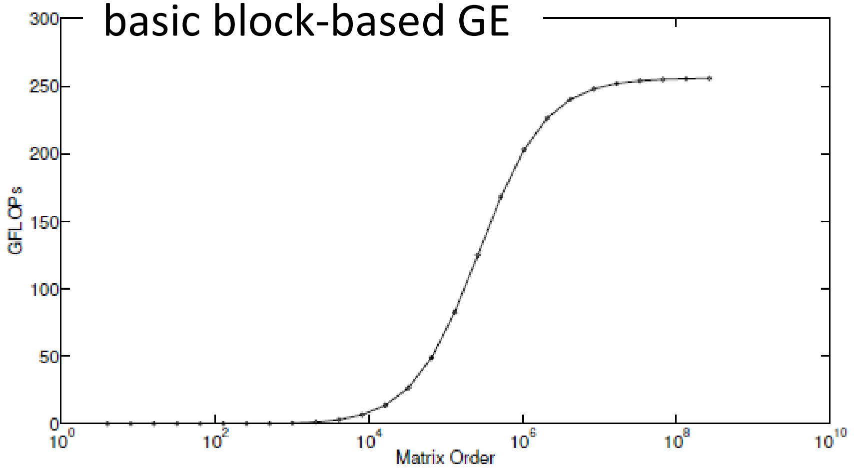
End Result (with other optimisations)

Limited by Slices (mainly due to pipeline registers to boost clock frequency)

	Resource Use		
Entity	ALMs (1000s)	DSPs	M20Ks
Individual PE	0.15	1	1
Stage 1	0.9	5	4
Stage 2	2.6	11	18
Stage 3	3.6	14	27
Stage 4	5.7	20	45
Stage 5	10	32	80
Stage 6	19	56	146
Stage 7	35	104	280
Stage 8	70	200	542
Stage 9	95	260	395
Full design	338	962	1932

Use more DSPs Use less memory

Achieve comparable performance to the peak possible using basic block-based GE



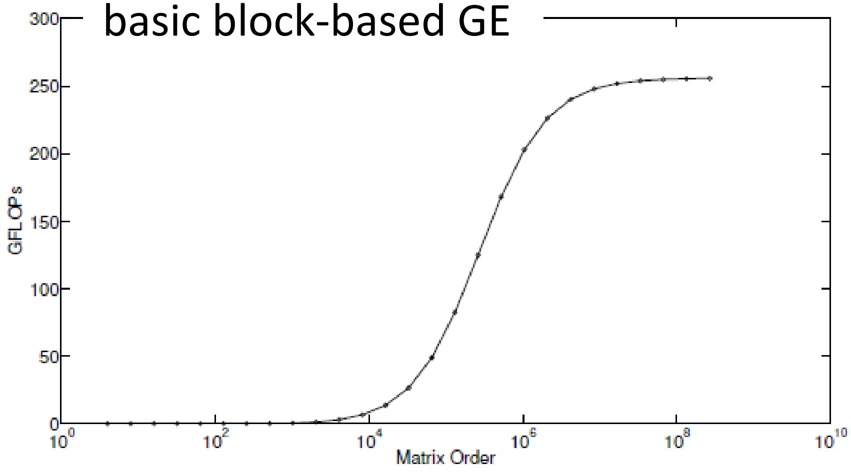
End Result (with other optimisations)

Limited by Slices (mainly due to pipeline registers to boost clock frequency)

	Resource Use		
Entity	ALMs (1000s)	DSPs	M20Ks
Individual PE	0.15	1	1
Stage 1	0.9	5	4
Stage 2	2.6	11	18
Stage 3	3.6	14	27
Stage 4	5.7	20	45
Stage 5	10	32	80
Stage 6	19	56	146
Stage 7	35	104	280
Stage 8	70	200	542
Stage 9	95	260	395
Full design	338	962	1932

Use more DSPs   Use less memory

Achieve comparable performance to the peak possible using basic block-based GE



Vs basic block-based GE, it will work on many more algorithms (subject to single precision being sufficient)

Summary

- The approach of this paper saves memory, achieves high performance and is numerically stable (and opens doors for some room for improvement)

Conclusions

- Please, please, please, no more GE/LU decomposition papers that don’t include some form of pivoting.
- Perhaps consider if its possible to re-examine the use of systolic arrays in your designs, perhaps with re-ordered inputs, to reduce I/O or memory.

Conclusions

- Please, please, please, no more GE/LU decomposition papers that don’t include some form of pivoting.
- Perhaps consider if its possible to re-examine the use of systolic arrays in your designs, perhaps with re-ordered inputs, to reduce I/O or memory.
- (Feel free to give me a Stratix 10 to see some big performance gains)

Conclusions

- Please, please, please, no more GE/LU decomposition papers that don't include some form of pivoting.
- Perhaps consider if its possible to re-examine the use of systolic arrays in your designs, perhaps with re-ordered inputs, to reduce I/O or memory.

- (Feel free to give me a Stratix 10 to see some big performance gains)
- (Don't want to offend anyone from Xilinx, I'll take your boards too)